



2.4 Speicherarchitektur

Peter Marwedel

Informatik 12

Otto-Hahn-Str. 16

Tel. 755 6111

E-mail: peter.marwedel@tu-dortmund.de

Sprechstunde: Mo 13:00-14:00

2013/09/03

Kontext

Prozessor

Leitwerk

Rechenwerk

Speicher-
architektur

Externe Kommunikation

Interne Kommunikation

Die Wissenschaft Informatik befasst sich mit der Darstellung, Speicherung, Übertragung und Verarbeitung von Information. [Gesellschaft für Informatik]

Entwurfsziele

- Möglichst große Kapazität
- Möglichst kleine Zugriffszeiten
  - Geringe Zeit bis zur Verfügbarkeit eines Speicherinhalts (kleine *latency*)
  - Kleiner zeitlicher Abstand zwischen Übertragungen, hoher Durchsatz (großer *throughput*)
- Persistente (nicht-flüchtige) Speicherung
- Geringe Energieaufnahme
- Hohe Datensicherheit
- Geringer Preis
- Physikalisch klein

Konflikt für <u>einen</u> Speicher:

Entweder hohe Kapazität oder schnell und energieeffizient

16 bit lesen; Größe in Bytes;
65 nm für SRAM, 80 nm für DRAM

SRAM

DRAM

40
35
30
25
20
15
10
5
0

512 2K 8K 32K 128K 512K 2M 8M 32M 128M 512M 2G

High Performance - 16 banks SPM Access time (ns)

High Performance - 16 banks SPM Energy (nJ) - read

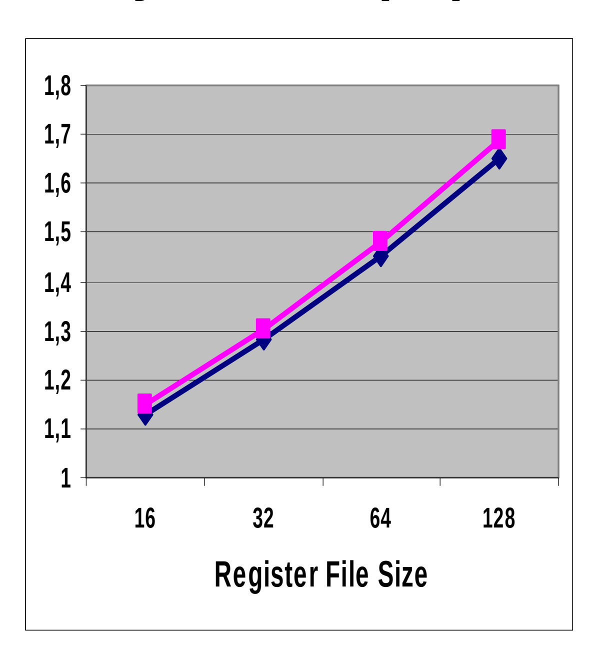
High Performance - 16 banks DDR Access time (ns)

High Performance - 16 banks DDR Energy (nJ) - read

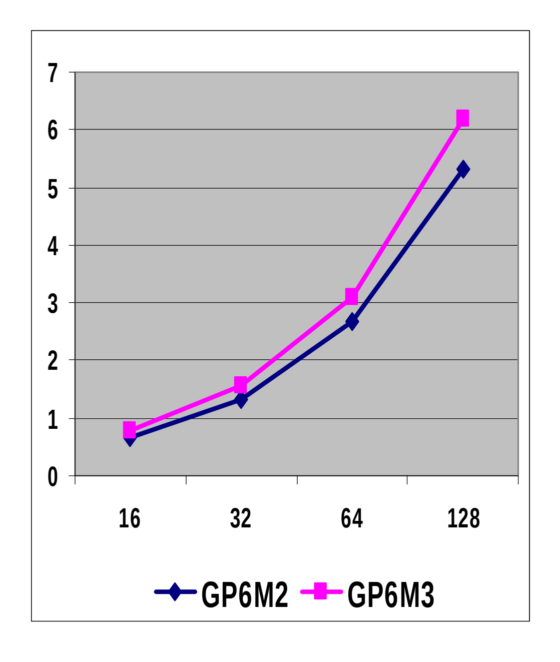
Quelle: Olivera Jovanovic, TU Dortmund, 2011

„Alles“ ist groß für große Speicher

Zykluszeit (ns)*



Fläche (λ^2x10^6)



El. Leistung (W)



* Monolithic register file; Rixner's et al. model [HPCA'00], Technology of 0.18 mm; VLIW configurations for a certain number of ports („GP*x*M*y*REG*z* where: x={6}, y={2, 3} and z={16, 32, 64, 128“}; Based on slide by and ©: Harry Valero, U. Barcelona, 2001

Kosten/Mbyte und Zugriffszeiten für verschiedene Technologien

SRAM (chip)
DRAM (chip
Disk

Cost ($/MB)
100,000
10,000
1,000
100
10
1
0.1
0.01

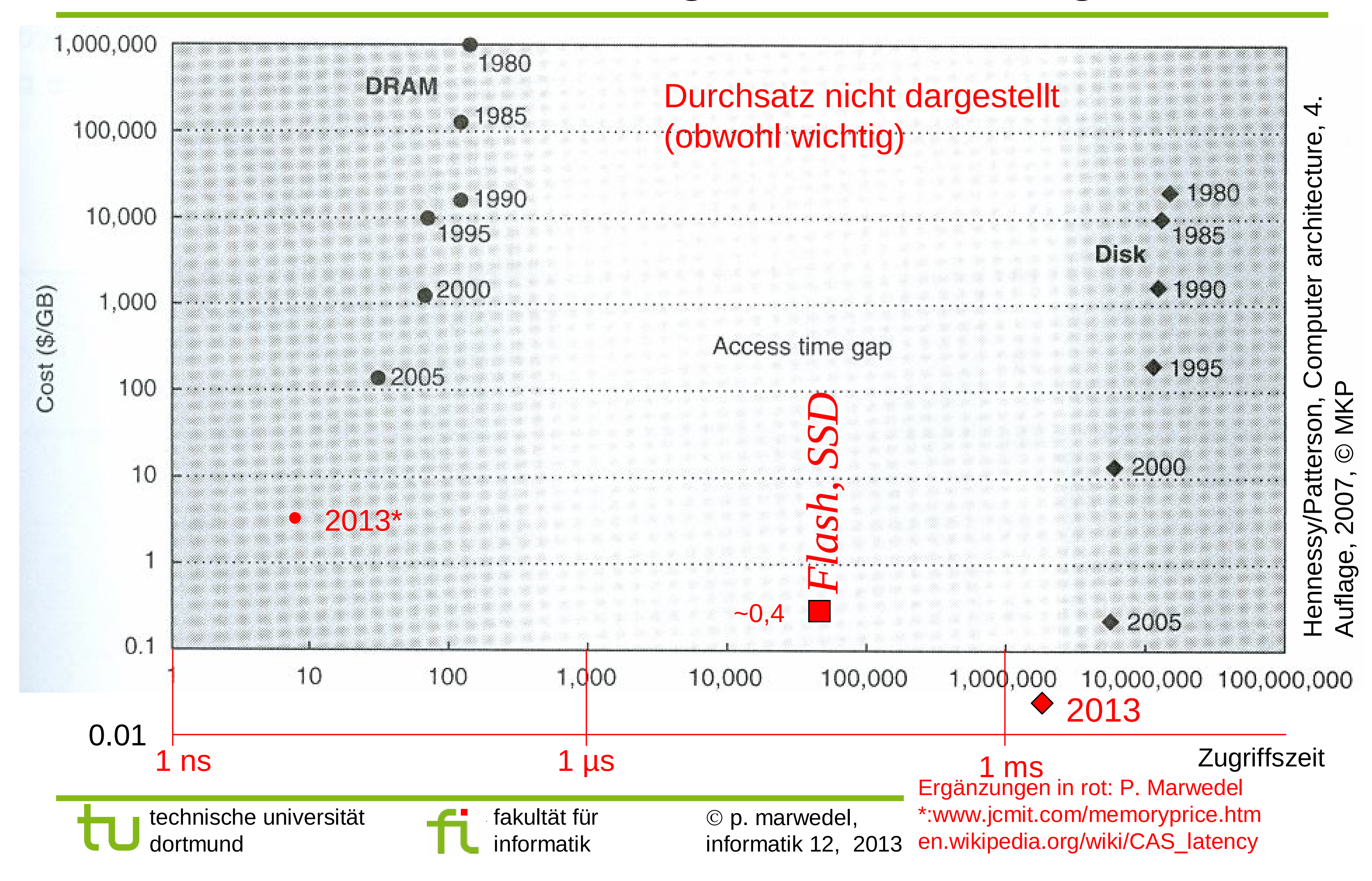
Access time (ns)
1
10
100
1,000
10,000
100,000
1,000,000
10,000,000
100,000,000

SRAM: 1980, 1985, 1990, 1995, 2000
DRAM: 1980, 1985, 1990, 1995, 2000
Disk: 1980, 1985, 1990, 1995, 2000

Access time gap

[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

Kosten/Mbyte und Zugriffszeiten für verschiedene verfügbare Technologien



Ergänzungen in rot: P. Marwedel
*:www.jcmit.com/memoryprice.htm
en.wikipedia.org/wiki/CAS_latency

Der Speicher-„Flaschenhals“ (*Memory wall*)

Geringe Steigerungsraten der Speicherzugriffsperformance

Speicherhierarchie

- Große Speicher sind langsam und benötigen viel Energie
- Anwendungen verhalten sich üblicherweise lokal
- ☞ Wir versuchen, häufig benötigte Speicherinhalte in kleinen Speichern, seltener benötigte Inhalte in großen Speichern abzulegen
- ☞ Einführung einer „Speicherhierarchie“
- Man möchte durch diese Kombination die Illusion eines großen Speichers mit kleinen Zugriffszeiten erzielen, zumindest im Mittel
- Eine enge Grenze für die Zugriffszeit ist vielfach kaum zu garantieren
- ☞ Bei Realzeitsystemen ergeben sich spezielle Überlegungen

Der Konflikt und die Idee sind nicht neu

“*Ideally one would desire an indefinitely large memory capacity such that any particular ... word ... would be immediately available - i.e. in a time which is ... shorter than the operation time of a fast electronic multiplier. ...*
It does not seem possible physically to achieve such a capacity.
We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.”

A.W. Burks, H.H. Goldstine, and J. von Neumann: Preliminary Discussion of the Logical Design of an Electronic Computing Element, **1946**

Mögliche Stufen der Speicherhierarchie
und derzeit eingesetzte Technologien

Tertiärspeicher (Optische Speicher, Bänder)

Sekundärspeicher (Platten, Flash)

Plattencaches (DRAM, Flash)

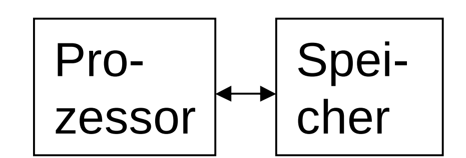
Haupt- bzw. Primärspeicher (DRAM)

Caches (SRAM) &TLBs

Register (SRAM)

persistent

Schlüssel zur Einführung der Hierarchie: Speicherverwaltung



Unterscheidung zwischen den Adressen, welche der Prozessor generiert und jenen, mit denen der physikalische Speicher adressiert wird.

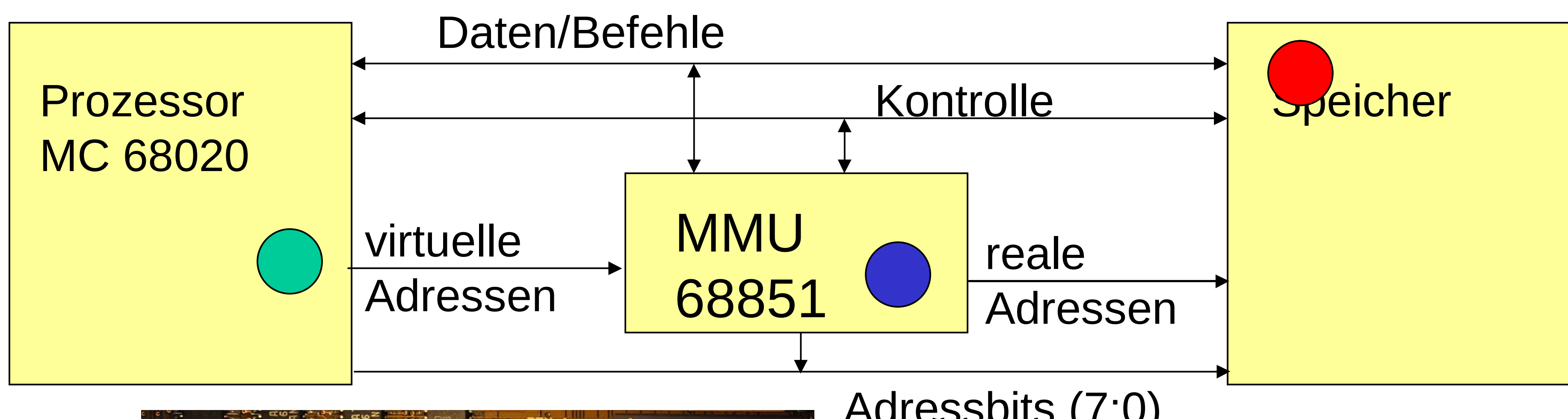
Def.: Prozessadressen bzw**. virtuelle Adressen** sind die effektiven Adressen nach Ausführung aller dem Assemblerprogrammierer sichtbaren Adressmodifikationen, d.h. die von einem Prozess dem Rechner angebotenen Adressen. *

Def.: Maschinenadressen, reale Adressen bzw**. physikalische Adressen** sind die zur Adressierung der realen Speicher verwendeten Adressen.

Memory Management Unit (MMU)

MMU ist für das Umrechnen der Adressen verantwortlich.

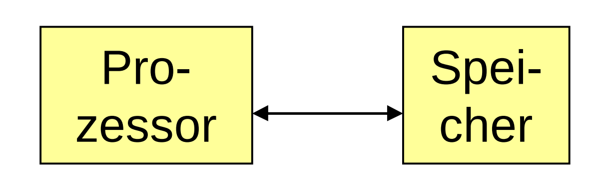
Beispiel Motorola



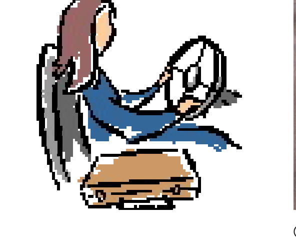


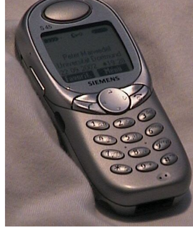
Identität

Bei einfachen Systemen sind reale und virtuelle Adressen identisch *(„memory management without swapping or paging“).*



Adressraum durch real vorhandenen Speicher beschränkt.

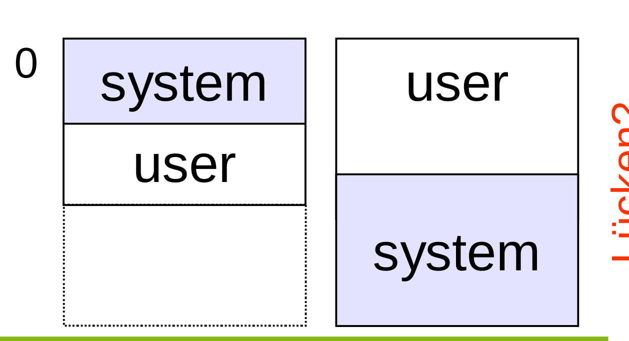




© P. Marwedel

Kommt v.a. bei eingebetteten Systemen (Fahrzeugelektronik, Handys usw.) vor, weniger bei PCs und Workstations.

SPIM/MARS



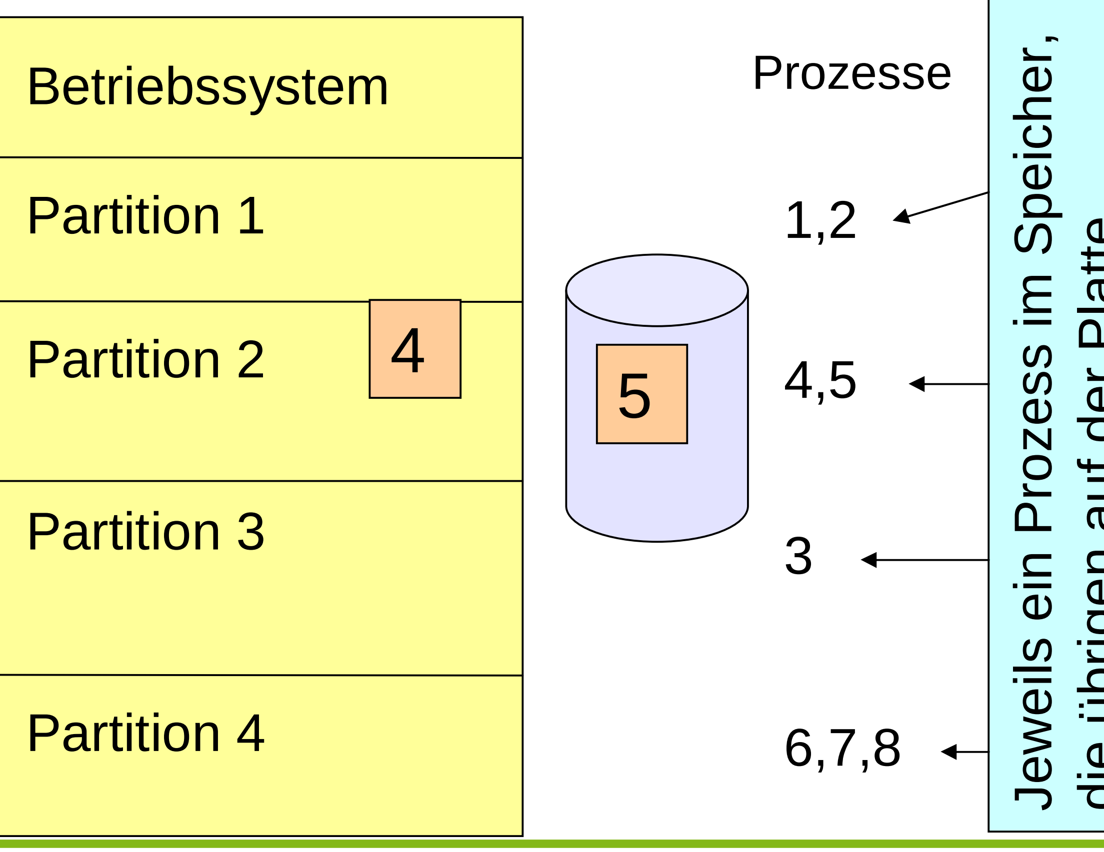
Sehr einfache Systeme:
- Aufteilung in Systembereich und Benutzerprogramm (☞ SPIM/MARS)

Einteilung in Laufbereiche (*core partitions*)

Aus Aufwandsgründen statt flexibler Speicherzuteilung an Prozesse feste Partitionierung des Speichers.

Adressbereich der Prozesse zusammen-hängend im virtuellen und im realen Adressraum.

Compiler adressieren Befehle ab Adresse 0. Beim Laden muss auf alle Adressen Anfangsadresse der Partition addiert werden. ☞ Identifikation von Adressen im übersetzten Programm!



Speicherschutz durch Grenzregister möglich

Z.Tl. nur Zugriff auf durch Grenzregister begrenzten Speicherbereich erlaubt (☞ Speicherschutz); Grenzregister werden bei *context switch* umgeladen.

Prozesse

1,2

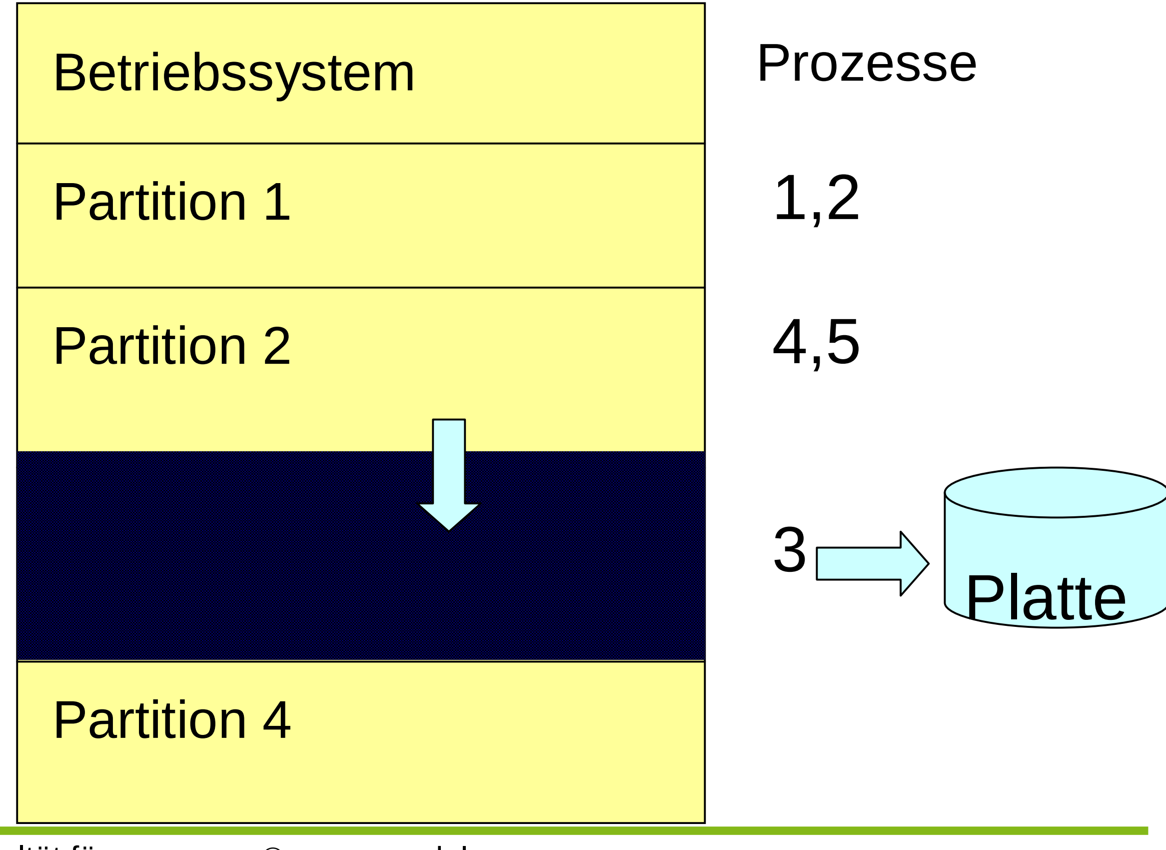
4,5

Partition 3 — 3

6,7,8

Was tun, wenn Prozesse mehr Speicher benötigen? (1)

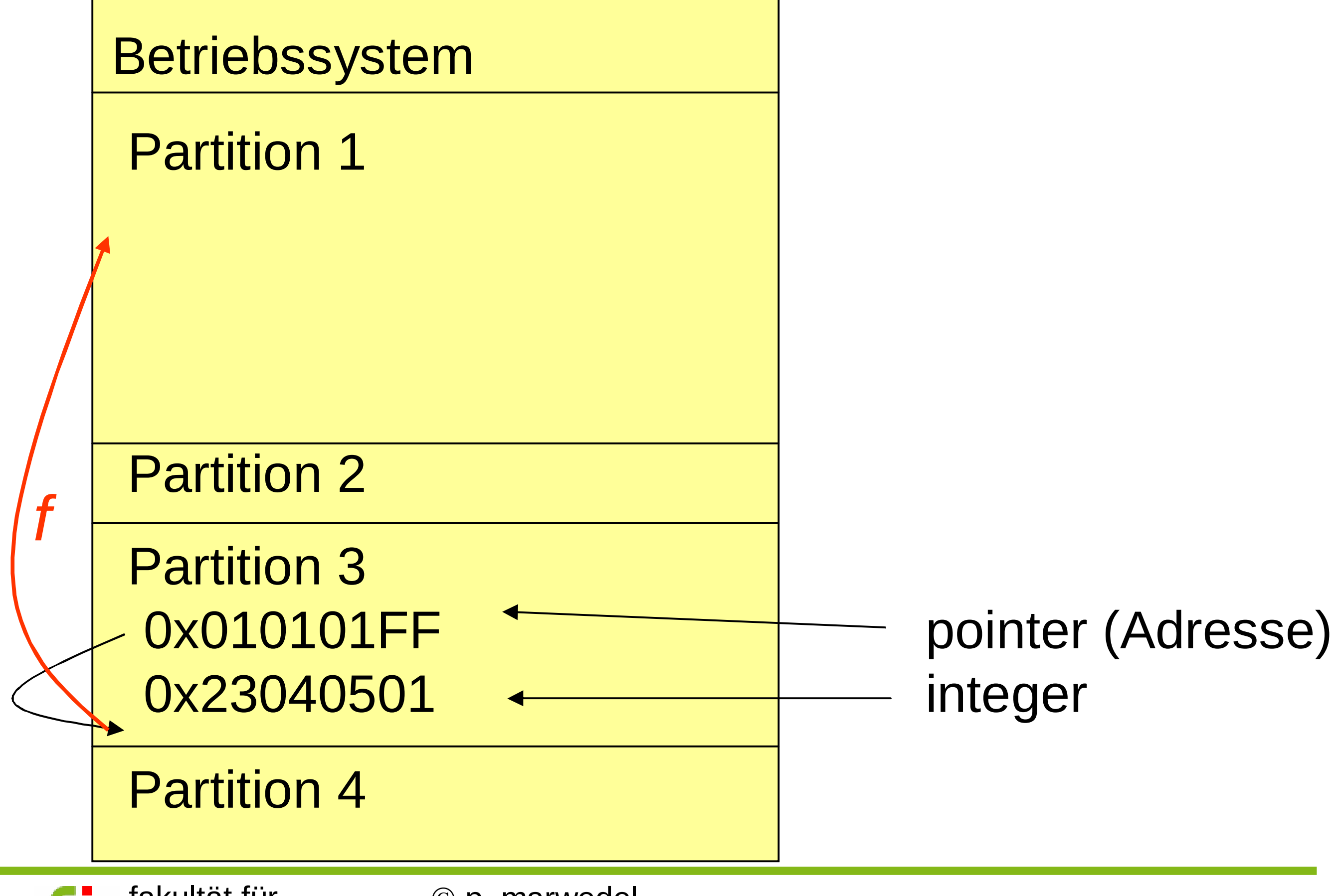
Man könnte Partitionen verschmelzen, wenn Prozesse mit dem Speicher der zugeteilten Partition nicht auskommen. Aus Aufwandsgründen selten realisiert.



Was tun, wenn Prozesse mehr Speicher benötigen? (2)

Verschieben von Prozessen im Speicher nicht möglich, da Adressen angepasst werden müssten, andere Informationen nicht.
Im Speicher sind Adressen und andere Informationen ununterscheidbar

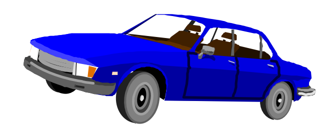
Aus demselben Grund kann nur der gesamte Inhalt einer Partition auf die Platte ausgelagert werden (***swapping***) und an dieselbe Stelle zurückgeschrieben werden.



Einsatzbereiche

Sehr einfache Rechner, eingebettete Systeme (Handys, Fahrzeugelektronik).
Systeme, bei denen die Programme von vornherein alle bekannt sind.





Beurteilung

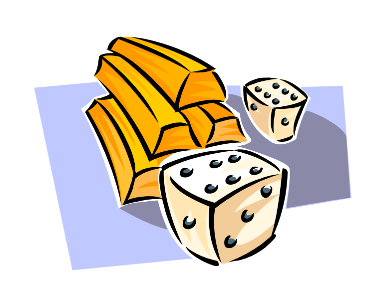


Nachteile:

Sehr starke „externe Fragmentierung“
(Speicherbereiche bleiben ungenutzt).

Adressraum durch real vorhandenen Speicher beschränkt.

Man muss den Speicherbedarf von Applikationen recht genau kennen, um eine möglichst passende Speichergröße zuzuweisen.



Risiko des Prozessabbruchs, wenn Speicher nicht ausreicht.

☞ Gründe für bessere Verfahren.

Seiten-Adressierung (*paging*)

- Einteilung der realen Adressen in Bereiche gleicher Länge (**Seitenrahmen, Kacheln, *page frames***).

Jede Seite kann auf jeder Kachel gespei-chert werden.

Kacheln

Seiten Prozess 1

Seiten Prozess 2

- Einteilung der virtuellen Adressen in Bereiche konstanter Länge (**Seiten, *pages***)

Seiten-Adressierung (*paging*)

Kacheln

Seiten Prozess 1

Seiten Prozess 2

virtuelle Adresse

Seitennr.	*Offset*

Zu jedem Prozess existiert eine Tabelle, die **Seitentabelle.**

P1:Seite	Kachel
0	1
1	5
2	7
3	6

reale Adresse = Anfangsadresse der Kachel + *Offset*

Vorteile

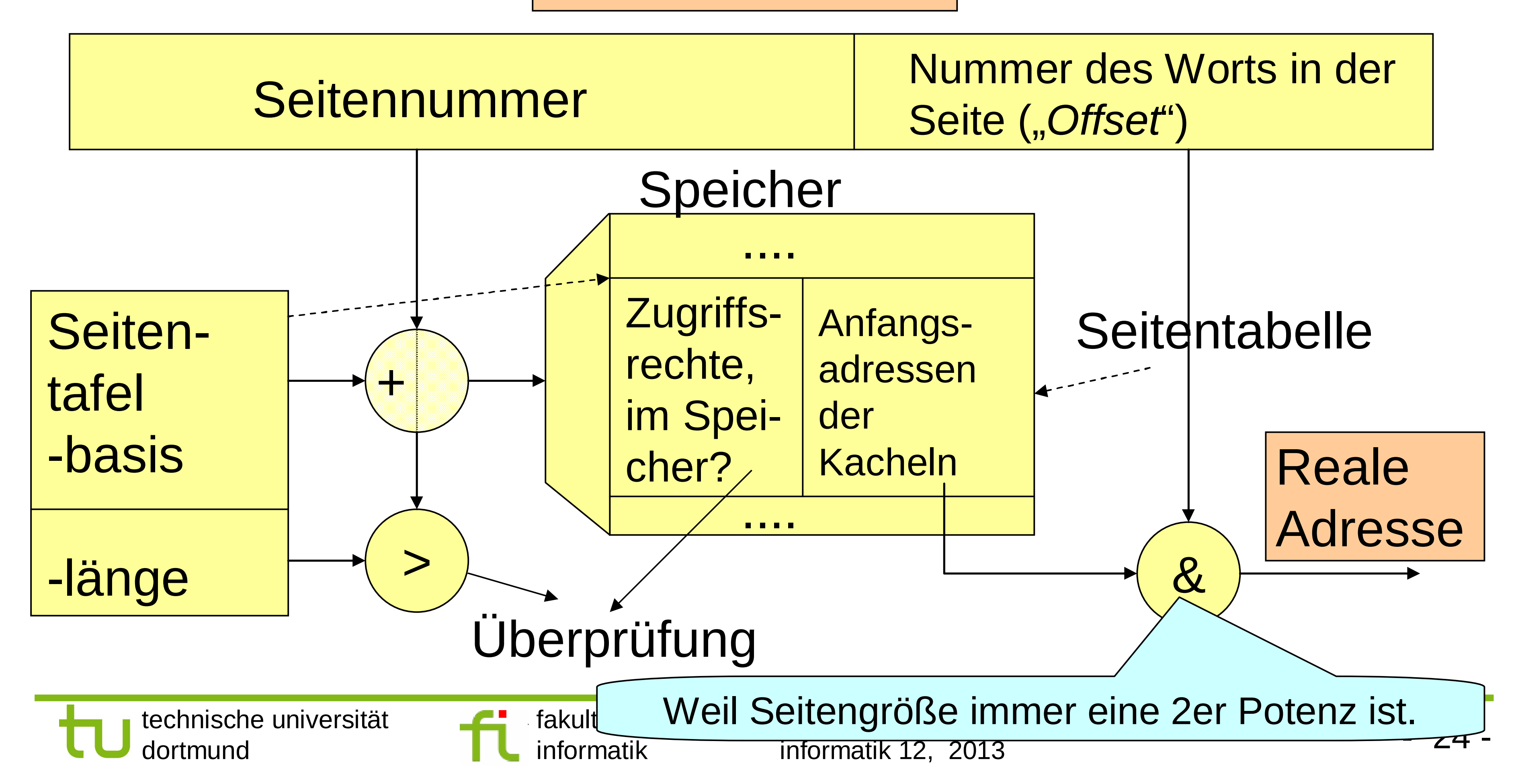
- Man muss sich nicht vorab überlegen, welchem Speicherbereich (*text, stack ,heap*) man wie viel Speicher zuweist.
- Jede Kachel des Speichers kann genutzt werden (keine „**externe Fragmentierung**").
- Lücken im virtuellen Adressraum (wie beim MARS/SPIM) sind kein Problem.
- Virtuelle Adressräume aller Prozesse können bei 0 beginnen, nur bei getrennt übersetzten Prozeduren sind Adressanpassungen erforderlich.

Nachteil (☹):

- „**interne Fragmentierung**"
  (Seiten nicht unbedingt gefüllt)

Organisation der Adressabbildung

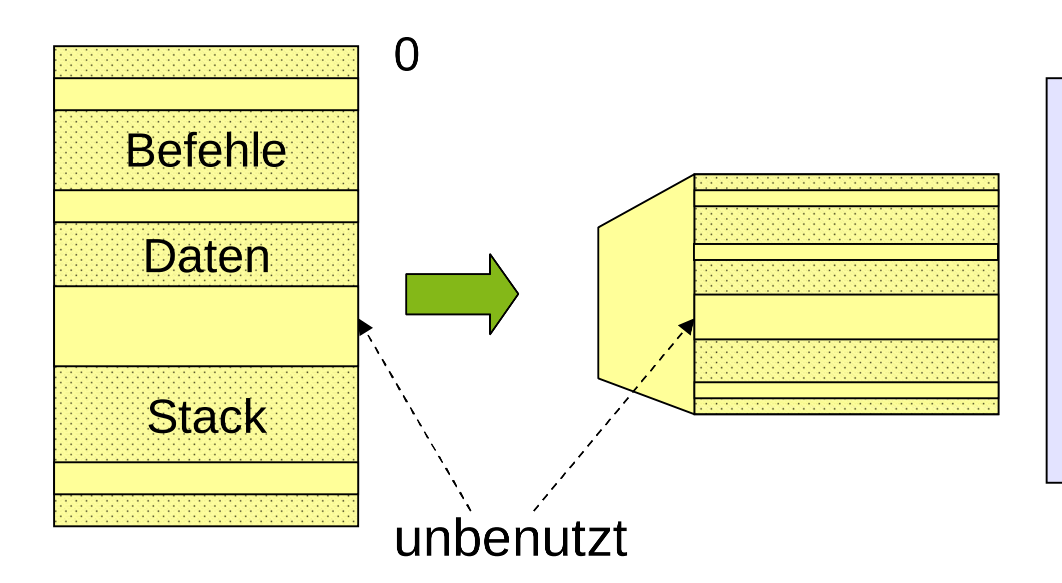
Schnelle Umrechnung von virtuellen auf reale Adressen!
Bei zusammenhängenden Adressen und Seitenadressierung:



Lücken im virtuellen Adressraum

Bei MIPS-Konvention: Adressbereiche zwischen Stack, Befehls- und Datenadressen werden meist nicht benutzt.

virtueller Adressraum Seitentabelle



Ungenutzte Seiten- bzw. Segmentnummern: bei der bisherigen Methode werden Tafeleinträge nicht benutzt!

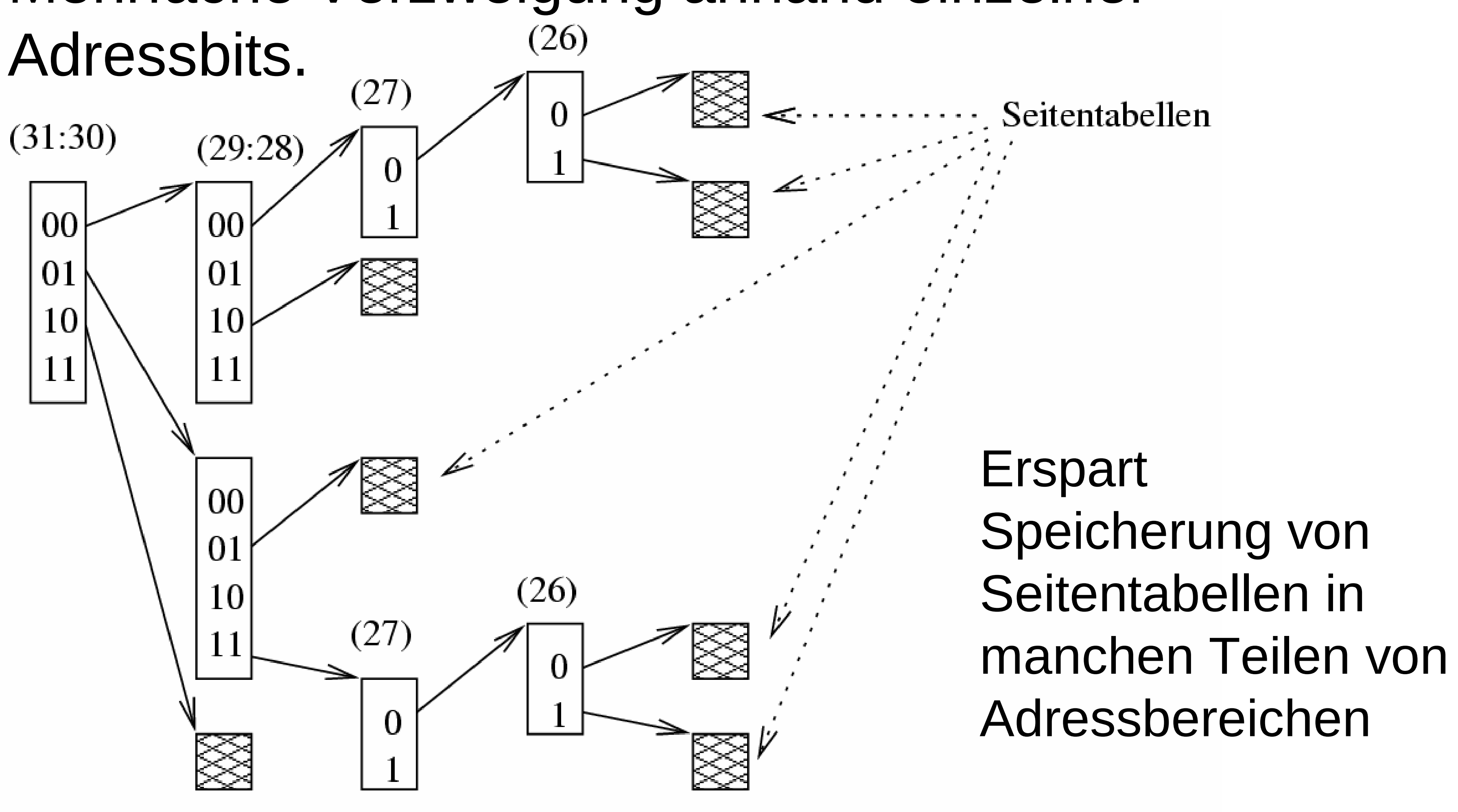
Seitentafel benötigt großen zusammenhäng. Speicherbereich.

☞ Berücksichtigung von Lücken im virtuellen Adressraum:

nicht für jede **mögliche** Seite auch Einträge in der Seitentafel!

Hierarchische Seitentabellen

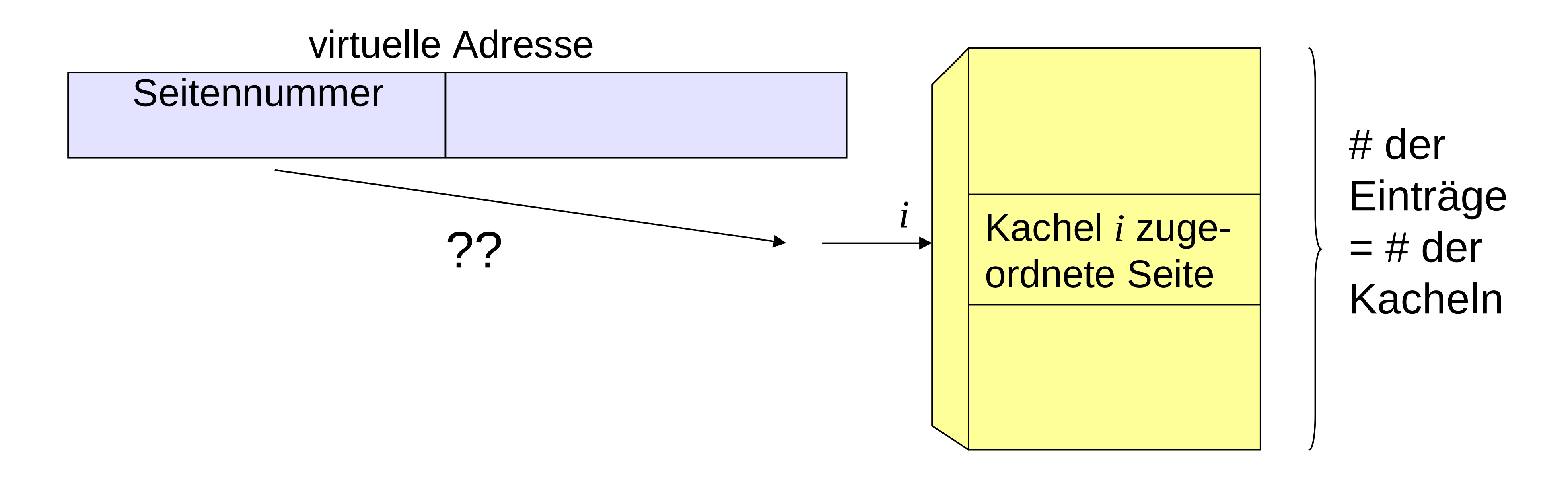
Mehrfache Verzweigung anhand einzelner Adressbits.



Erspart Speicherung von Seitentabellen in manchen Teilen von Adressbereichen

Invertierte Seitentabelle

Selbst bei hierarchischen Seitentabellen kann ein großer Speicherbedarf entstehen, v.a. bei großen Adressräumen
☞ „invertierte Seitentabellen“



Suche durch die gesamte Kacheltabelle wäre zu langsam

Beschleunigung mittels Hashtabellen

virtuelle Adresse

Seitennummer: Tag | Index	

Lineare Ketten von Zeigern auf Einträge in der invertierten Seitentabelle, die zu einem bestimmten Seitenindex gehören

Hash-tabelle

Kachel i zugeordnete Seite

i

der Einträge = # der Kacheln

Sequentielle Suche nach passendem Tag in der invertierten Seitentabelle; kein Treffer, falls Seite nicht im Speicher.

Tradeoff: Laufzeit ⇔ Speicherplatz

Zusammenfassung



- Viele Rechensysteme unterstützen *Multitasking*
- Reale und virtuelle Adressen, *memory management*
- Formen der Speicherverwaltung
  - Identität
  - Seitenadressierung
    - Sharing
  - Segmentadressierung
  - Segmentadressierung mit Seitenadressierung

Sharing (*shared libraries*)



Einfache Idee: Seiten-Abbildungen mehrerer Prozesse führen zur selben Kachel

virt. Adressen Prozess A

Xlib

0
1
2
3
4

Seitentabelle Prozess A

realer Speicher

j ? & …

0
1 Xlib
2
3
4
5
6
7
.
.
.

virt. Adressen Prozess B

Xlib

0
1
2
3
4

Seitentabelle Prozess B

Idee ist so nicht realisierbar

- Im gemeinsam genutzten Code kommen (virtuelle) Befehlsadressen vor, die sich z.B. auf Adressen in diesem Code beziehen.
- Adressen müssen Seitennummern verwenden, die nach Abbildung auf die realen Adressen über die jeweiligen Seitentafeln wieder auf den gemeinsam genutzten Bereich verweisen.
- Wenn diese Seitennummern ≠ sind: ☞ nicht im Code eintragbar (in 1 Speicherzelle nur 1 Seitennummer).
- **Nutzen Prozesse gemeinsamen Code, so muss dieser bei der Seitenadressierung bei allen beteiligten Prozessen auf dieselben virtuellen Adressen abgebildet werden.**

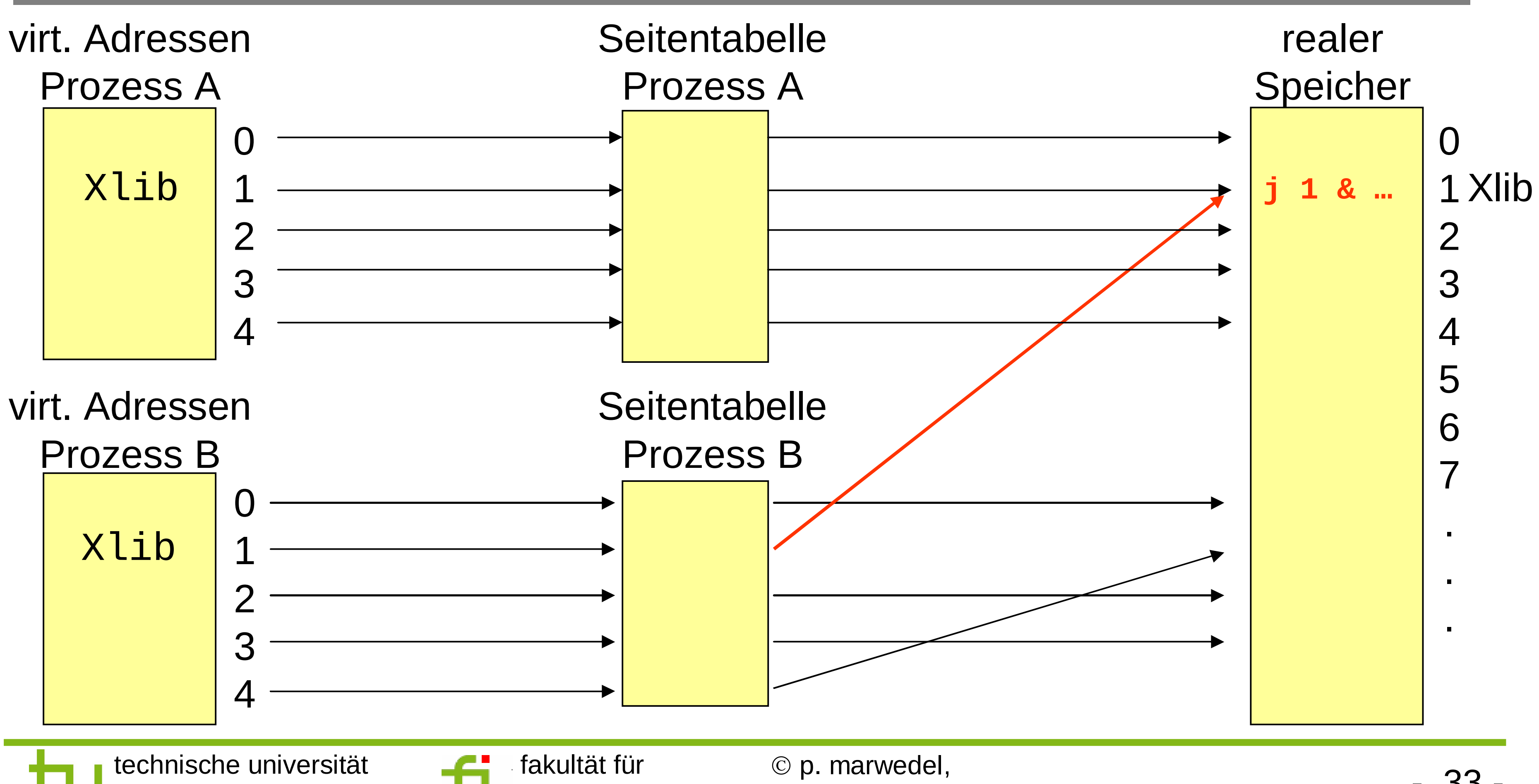
Idee ist so nicht realisierbar (2)

- ☞ **Nutzen Prozesse gemeinsamen Code, so muss dieser bei der Seitenadressierung bei allen beteiligten Prozessen auf dieselben virtuellen Adressen abgebildet werden.**
- ☞ Windows nutzt für gemeinsamen Code die obere Hälfte des Adressbereichs und benutzt Absprachen zur Adressvergabe.

?

Realisierbares *Sharing*

Shared libraries befinden sich an derselben Stelle des virtuellen Adressraums

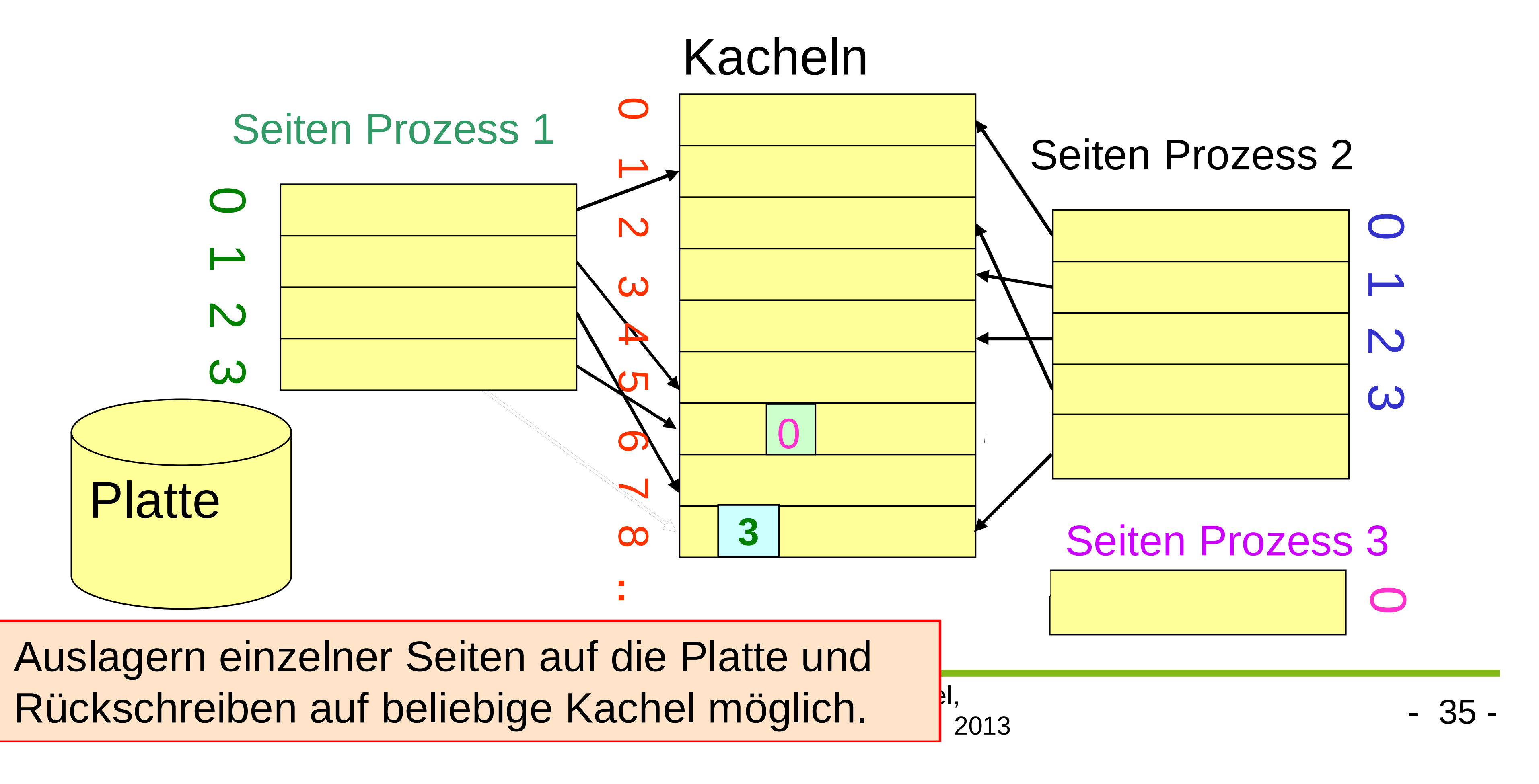


Beobachtungen

- *Paging* in der beschriebenen Form funktioniert ohne größeren Speicher (Plattenlaufwerk, *hard disc* (HD)) als weiteren Speicher.
- Wird bei Systemen ohne HD eingesetzt (in vielen eingebetteten Systemen).
- *Paging* bietet Form des **Speicherschutzes**: Prozesse können unterschiedliche Rechte zur Nutzung von Speicherbereichen haben.
- Speicherschutz mittels *paging* ist grob: Wunsch, durch verschiedene Prozesse auf Speicheradressen abgebildete Peripheriegräte mit unterschiedlichen Rechten zu nutzen. Häufig werden viele Peripheriegeräte auf dieselbe Seite abgebildet.

Demand paging (Realisierung einer Speicherhierarchie)

Wenn größerer und langsamerer Speicher (HD) verfügbar:
Einteilung des Speichers in Bereiche konstanter Länge + automatisches Auslagern selten benutzter Seiten sowie Einlagern von ausgelagerten Seiten, wenn auf die entsprechenden Adressen zugegriffen wird.

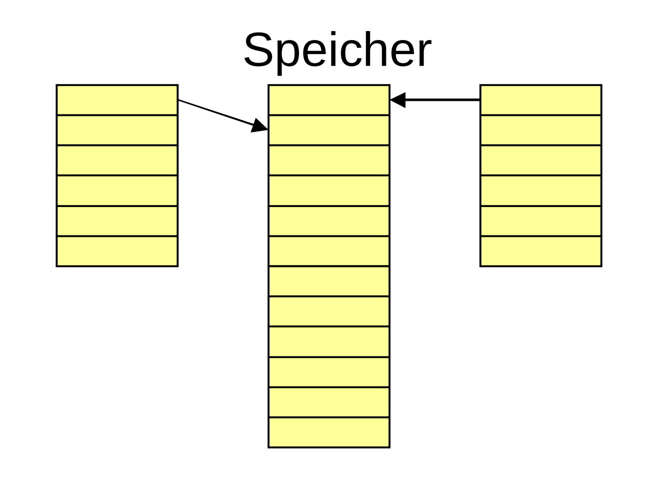


Auslagern einzelner Seiten auf die Platte und Rückschreiben auf beliebige Kachel möglich.

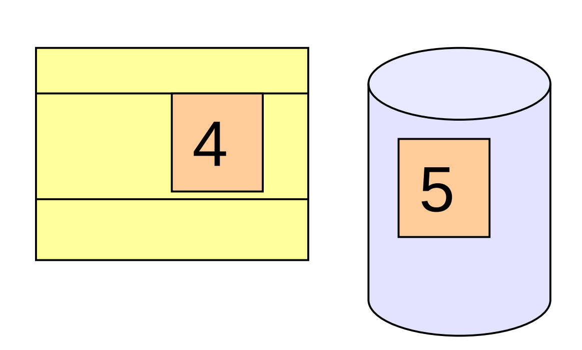
Paging

Paging ist mehrdeutig:

1. Einteilung des Speichers in Bereiche konstanter Länge, evtl. ohne Ein- und Auslagern (z.B., weil man Realzeit-Verhalten erzielen will).



2. Benutzung im Sinne von *demand paging.*




Definition 2 ist die Gebräuchlichere.

Segmentadressierung (1)

Zusammenhängenden Speicherbereichen (für Befehle, Daten, und den *stack*) im Prozessadressenraum werden **zusammenhängende Bereiche im realen Adressraum** zugeordnet.

Beispiel:

Segment 1 (Befehle Prozess 1, schreibgeschützt)
Segment 2 (*Stack* Prozess 2)
frei
Segment 4 (Befehle Prozess 2)
Segment 3 (Daten Prozess 1)
S. 5, *Stack* Proz.1
S. 6, Daten Proz.2

Segmentadressierung (2)

Beispiel:

virtuell	real	Speicher
0	0100	Segment 1 (Befehle Prozess 1, schreibgeschützt)
0	0230	Segment 2 (*Stack* Prozess 2)
	0304	frei
0	0390	Segment 4 (Befehle Prozess 2)
0	0520	Segment 3 (Daten Prozess 1)
0	0720	S. 5, *Stack* Proz.1
0	0830	S. 6, Daten Proz.2

Prozesse adressieren in jedem Segment ab Adresse 0;

Zu jedem Prozess gehört eine Tabelle der für ihre Segmente zu addierenden Adressen (**verdeckte Basen**).

Prozess 2		
Befehle	0390	
Daten	0830	
Stack	0230	

Länge

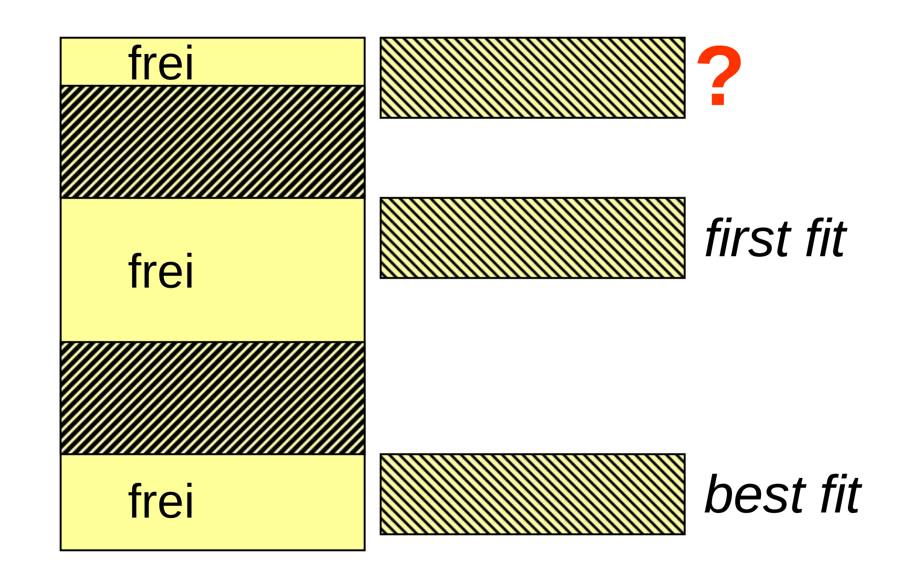
Erfordert Befehlssatz mit separaten Befehlen für *stack*-Zugriffe

Bei jedem Speicherzugriff wird die verdeckte Basis des jeweiligen Segments zur virtuellen Adresse addiert.

Eigenschaften der Segmentadressierung (1)

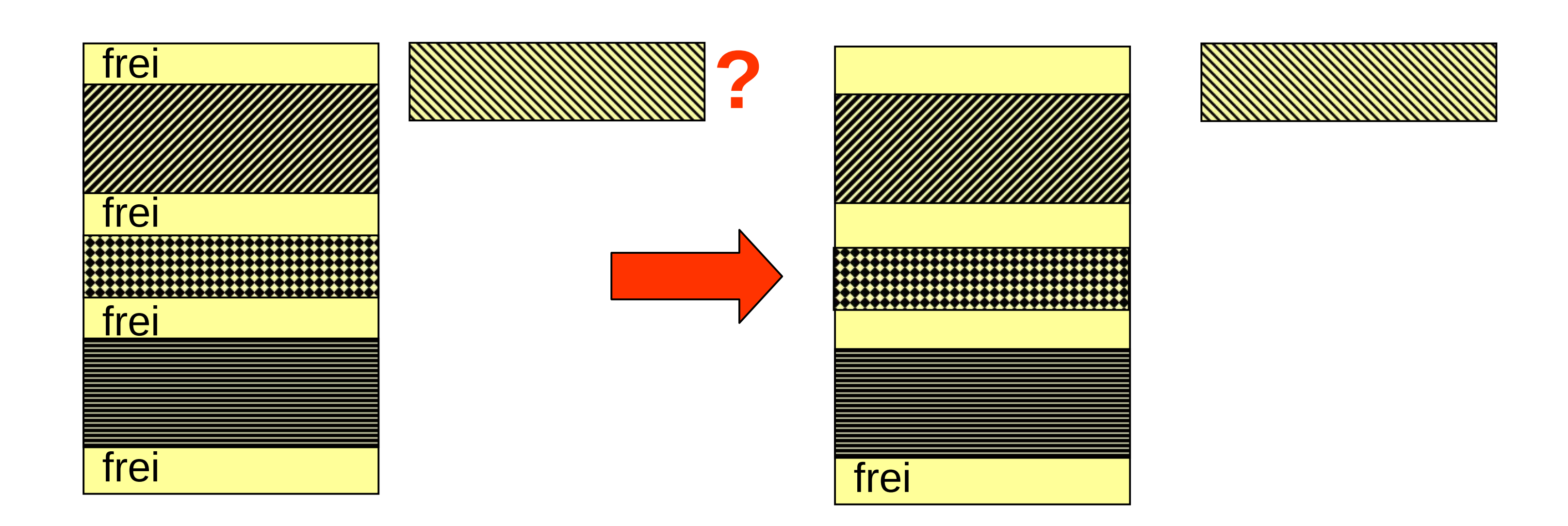
- Speicherschutz: Abfrage der Segmentlänge, R/W/E-Modi
- Beim Laden müssen keine Adressen angepasst werden, dies passiert stets zur Laufzeit.
- Verschieben von Segmenten ist möglich, da alle Adressen mittels Änderung der verdeckten Basis automatisch angepasst werden.
- Auslagern ganzer Segmente und Rückschreiben an beliebige, ausreichend große Lücke ist möglich.

Strategien zur Suche einer ausreichend großen, freien Lücke:



Eigenschaften der Segmentadressierung (2)

- Segmente können sich ausdehnen. Ist der anschließende Speicher belegt, so müssen sie umkopiert werden.
- Ein Teil des Speichers außerhalb der Segmente bleibt unbenutzt (externe Fragmentierung).
- Ggf. müssen Segmente verschoben werden, um ausreichend große freie Bereiche herzustellen.



Adressabbildung bei Segmentadressierung

Bei zusammenhängenden Segmentnummern effizient.

2 Formen der Segmentadressierung

- Segmentnummern in separaten Registern
  Beispiel: Intel Pentium
  Kein Problem mit dem *sharing:* Segmentregister müssen nur auf den gemeinsam genutzten Speicher „zeigen“.
  Wird von Microsoft nicht genutzt.
- Segmentnummern sind Teil der Adressen im Speicher
  Beispiel: PowerPC, IBM-Großrechner
  Beim *sharing* dasselbe Problem wie bei Seitenadressierung

Kombination von Segment- und Seitenadressierung

Segmentadressierung besitzt:

- Vorteile beim *Sharing*, wenn Segmentnummern in Registern gespeichert werden.
- Nachteile durch externe Fragmentierung

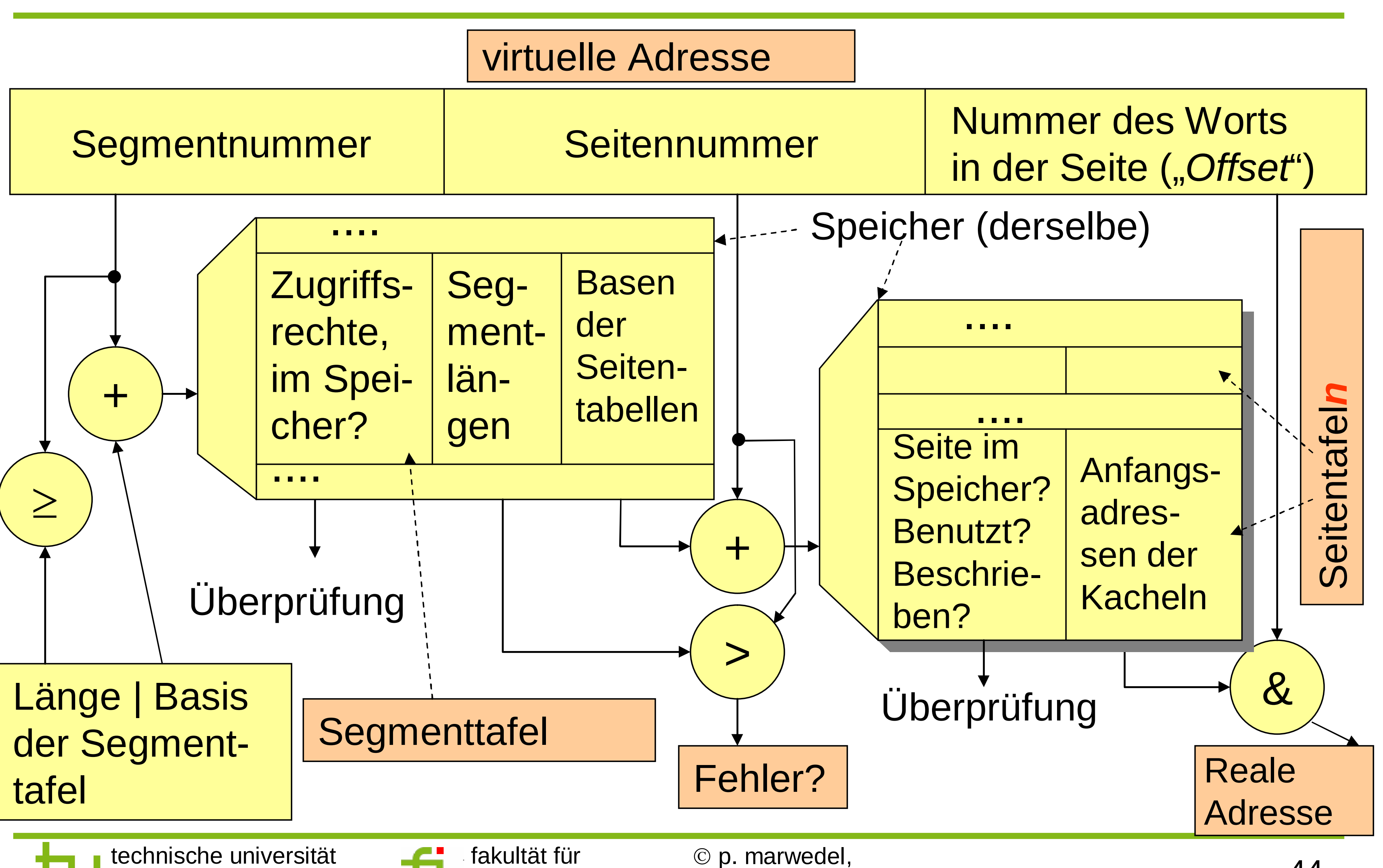
☞ **Kombination von Segment- und Seitenadressierung**:

- Jedes Segment enthält Seiten, diese wiederum Worte
- Fragmentierung reduziert sich auf interne Fragmentierung
- Jede Kachel kann jede Seite aufnehmen (kein Kopieren)
- Vorteile bei *shared libraries*

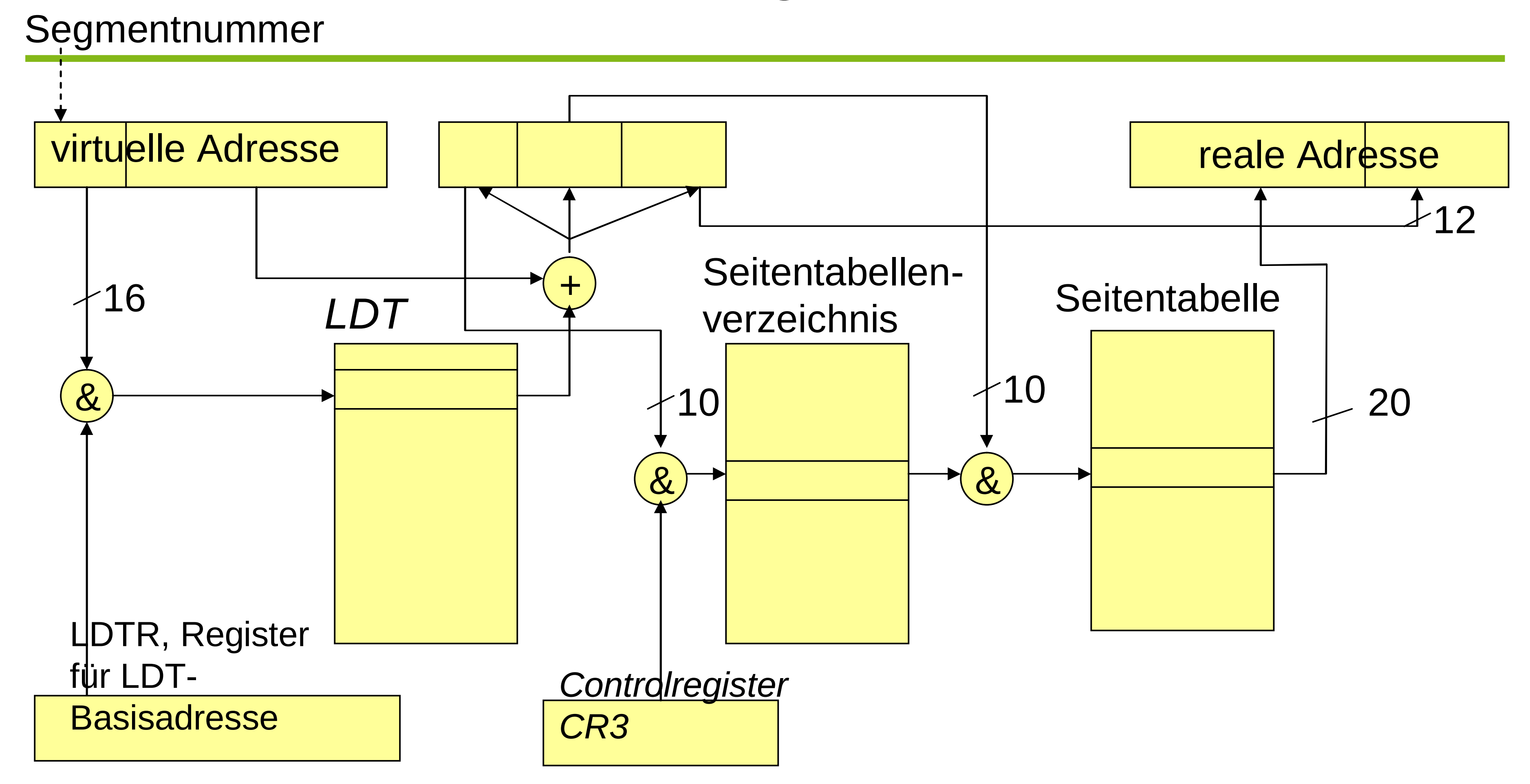
virtuelle Adressen

Segmentnr.	Seitennr.	Offset

Segmentadressierung mit Seitenadressierung



Adressabbildung beim Intel 386



Die *Local Descriptor Table LDT* benötigt Einträge für den Bereich möglicher Segmentnummern; Tabellen für unbenutzte Seiten sind nicht erforderlich. Diese haben selbst wieder die Größe einer Seite. Adressübersetzungstabellen können teilweise auf die Platte ausgelagert werden

Zusammenfassung



- Viele Rechensysteme unterstützen *Multitasking*
- Reale und virtuelle Adressen, *memory management*
- Formen der Speicherverwaltung
  - Identität
  - Seitenadressierung
    - Sharing
  - Segmentadressierung
  - Segmentadressierung mit Seitenadressierung

Mögliche Stufen der Speicherhierarchie
und derzeit eingesetzte Technologien

Tertiärspeicher (Optische Speicher, Bänder)

↕

Sekundärspeicher (Platten, Flash)

↕

Plattencaches (DRAM, Flash)

↕

Haupt- bzw. Primärspeicher (DRAM)

↕

Caches (SRAM) & TLBs

↕

Register (SRAM)

persistent (Tertiärspeicher, Sekundärspeicher)

2.4.3 *Translation Look-Aside Buffer (TLBs)*

Seitentabellen und Segmenttabellen im Hauptspeicher:

- ☞ Jeder einzelne Zugriff auf einen Befehl oder ein Datum erfordert zusätzliche Speicherzugriffe zur Umrechnung von virtuellen in reale Adressen.
- ☞ Unakzeptable Verlangsamung der Ausführung.
- ☞ Einführung zusätzlicher, kleiner, **schneller** Hardware-speicher, welche häufig die benötigten Tabelleneinträge enthalten.

Diese heißen ***translation look aside buffer (TLB)*** oder auch ***address translation memory (ATM).***

Die 3 Organisationsformen von TLBs

In jedem Fall enthalten die TLBs nur Auszüge aus den vollständigen Tabellen, die weiterhin im Speicher sind.

Ist die gesuchte Seiten- bzw. Segmentnummer nicht im TLB enthalten, so muss in der vollständigen Tabelle nachgesehen werden.

Der Eintrag aus der vollständigen Tabelle verdrängt dann einen Eintrag aus dem TLB nach einem noch zu bestimmenden Algorithmus.

Drei Organisationsformen von TLBs:

1. ***direct mapping***,
2. **Mengen-assoziative Abbildung** *(set associative mapping)*,
3. und **Assoziativspeicher *(associative mapping)***.

Direct Mapping

Die Seitennummer oder ein Teil davon adressiert den TLB.

virtuelle Adresse

Seitennummer		Nummer des Worts in der Seite („*Offset*“)
Tag	Index	

TLB

Tags	Zugriffs-rechte, im Spei-cher?	Anfangs-adressen der Kacheln

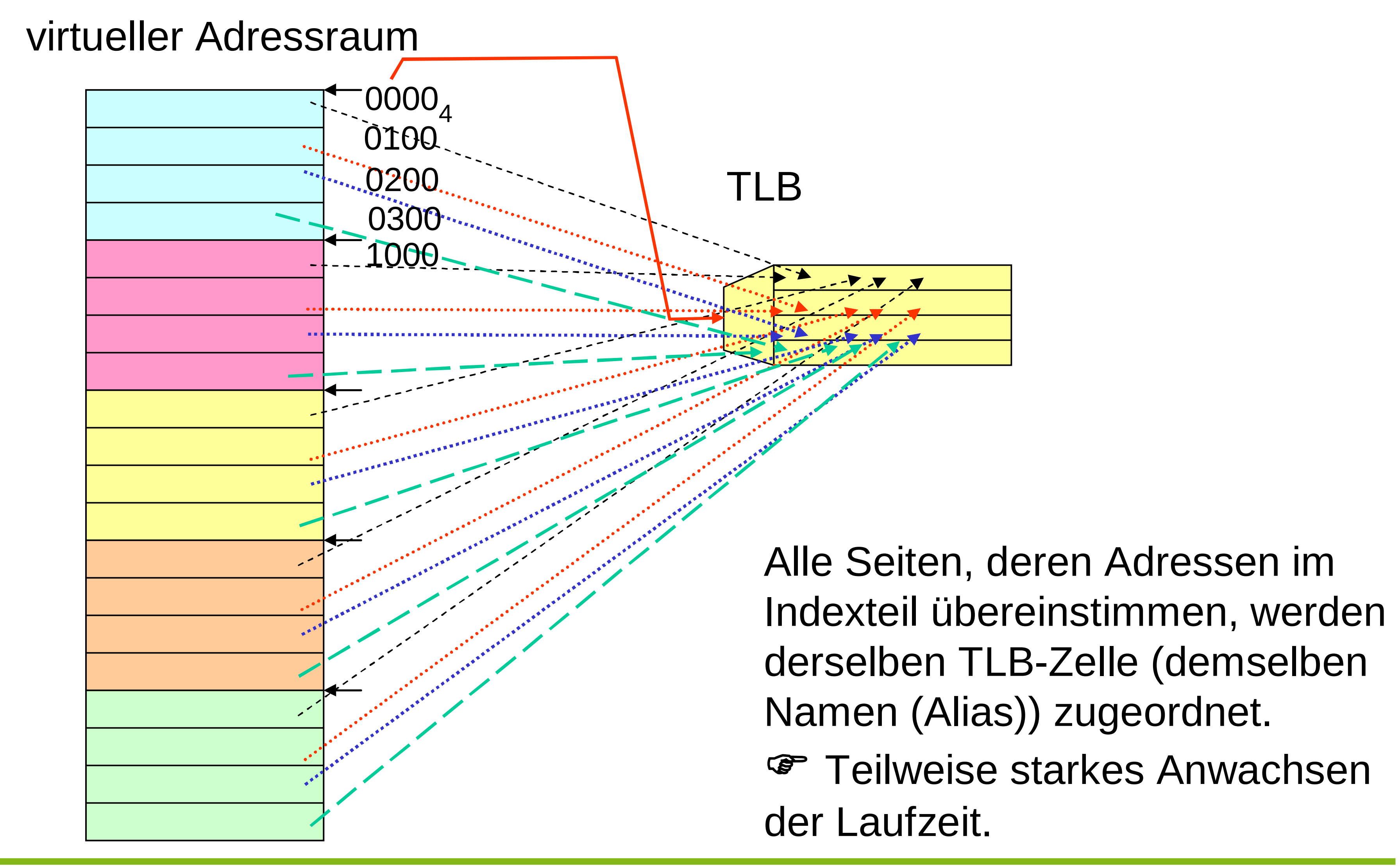
=

&

Reale Adresse

Tags gleich?

Aliasing bei *direct mapping*



Alle Seiten, deren Adressen im Indexteil übereinstimmen, werden derselben TLB-Zelle (demselben Namen (Alias)) zugeordnet.

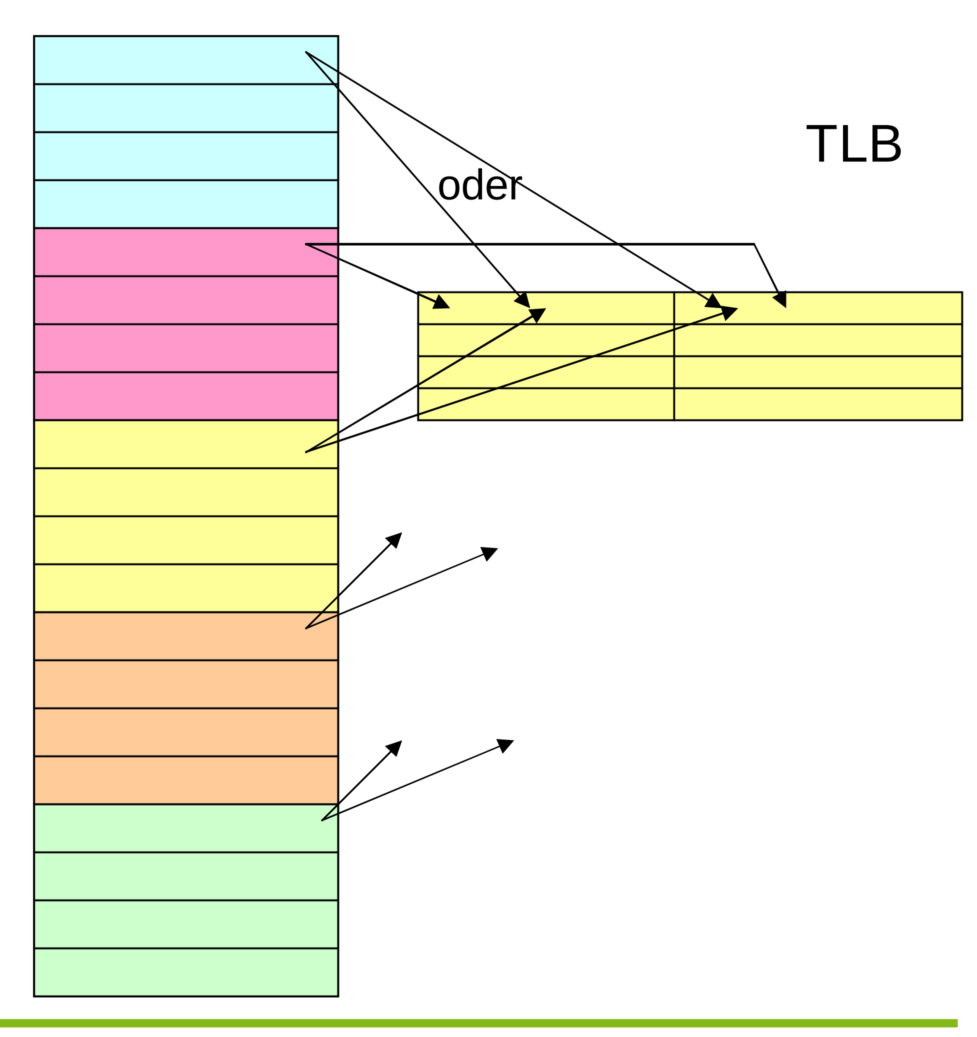
☞ Teilweise starkes Anwachsen der Laufzeit.

Mengen-assoziative Abbildung: Prinzip
n-way set associative mapping

Zu jedem Index-Wert gibt es n zugeordnete Plätze, mit n>1;
n heißt **Assoziativität**
„*n ways*“ genannt

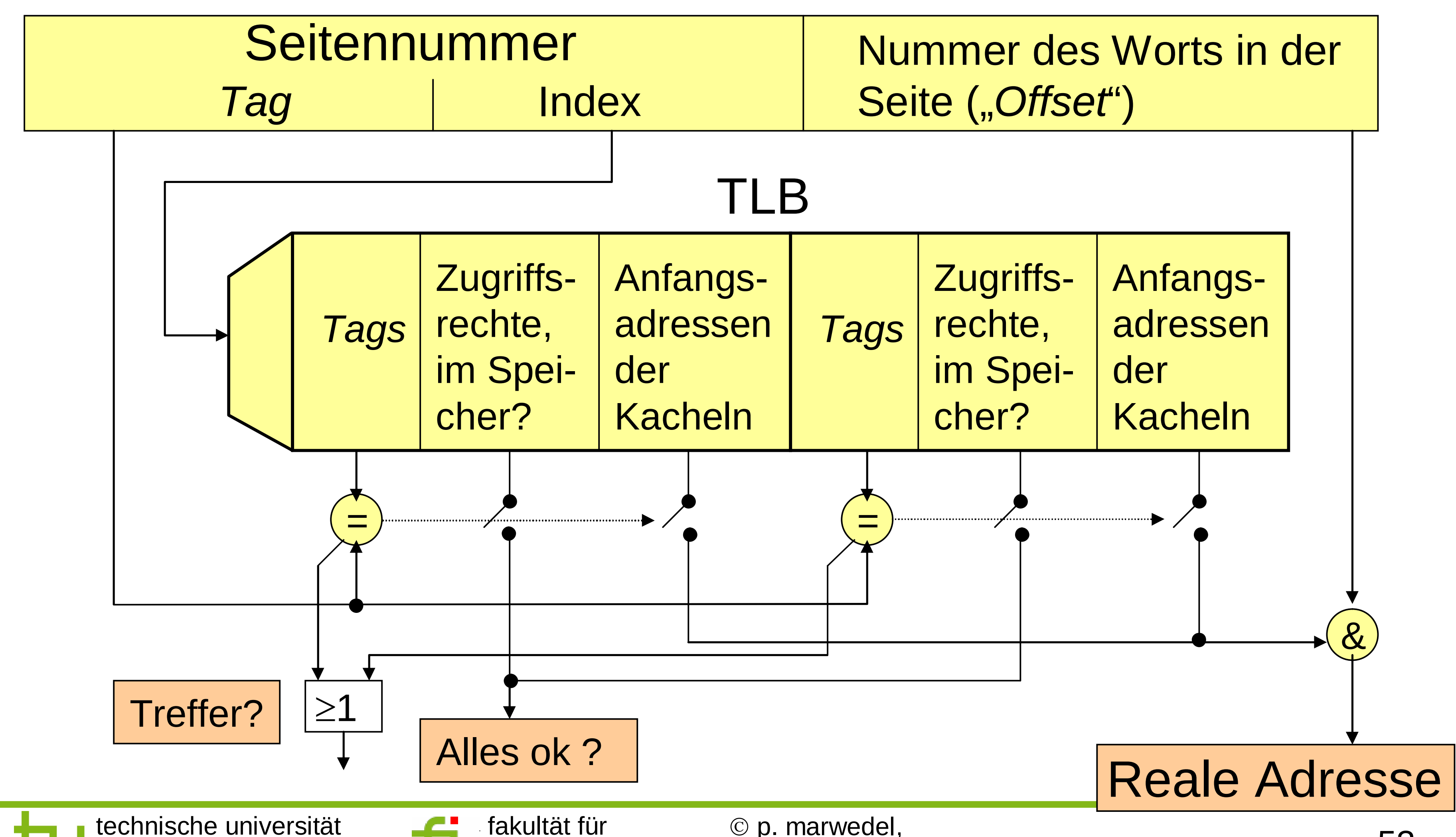
Wenn über die vollständigen Tabellen im Hauptspeicher abgebildet wird, so wird ein Platz im TLB überschrieben, z.B. der am Längsten nicht benutzte (*least recently used* (LRU)-Strategie)





Mengen-assoziative Abbildung
- Realisierung -

Beispiel: Setgrösse n=2



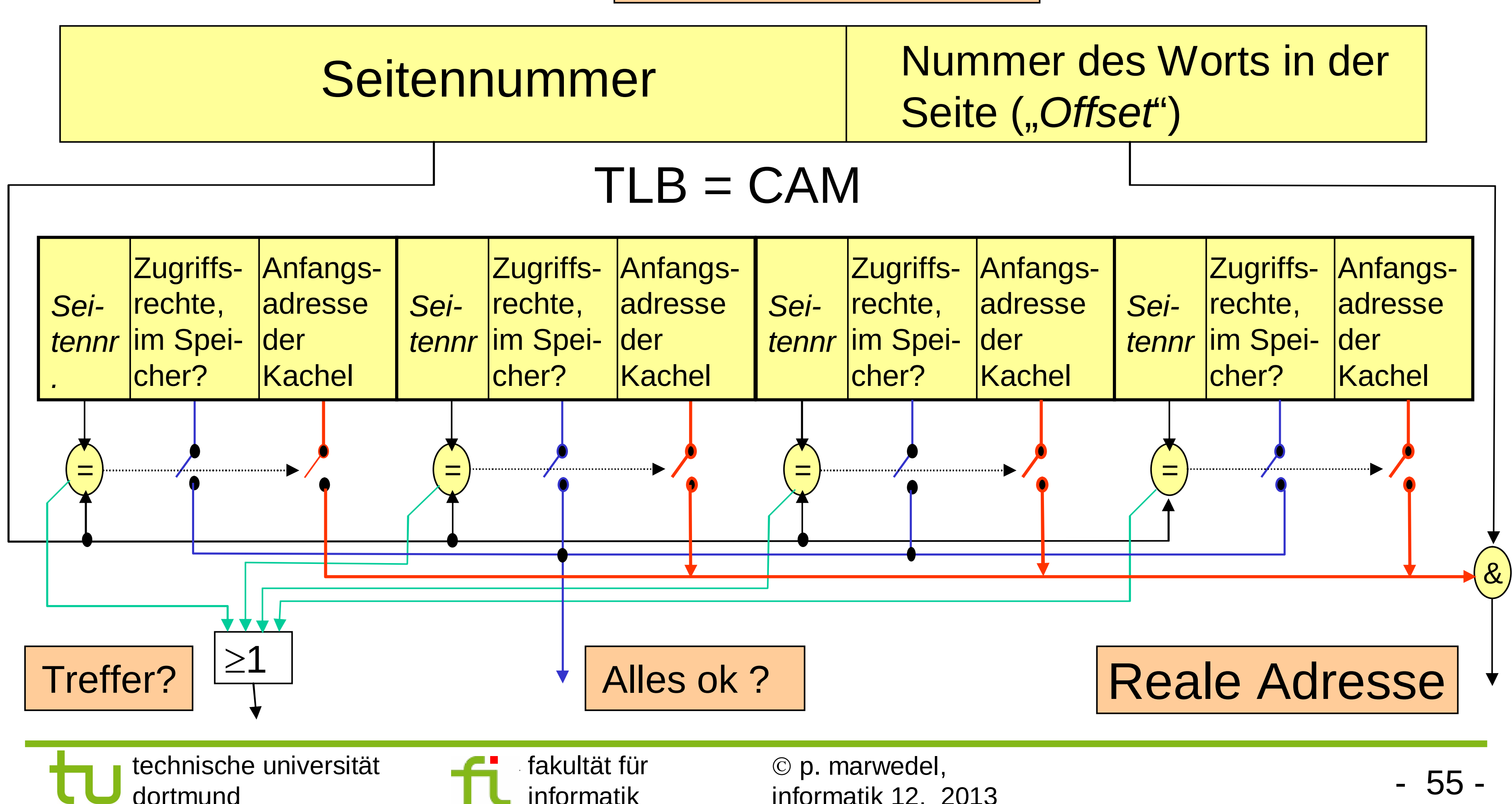
Assoziativspeicher, *associative mapping* - Prinzip -

Der Fall *associative mapping* entsteht aus dem *set associative mapping* für „Anzahl der Bits für den Index-Teil
→ 0“.

Jeder Platz des TLB kann die Verwaltungs-Information für jede beliebe Seite enthalten (kein *aliasing*).

Hardware-mäßige Realisierung mit ***content-adressable memory (CAM).***

Ein CAM vergleicht die Seitennummern aller Zellen gleichzeitig mit dem angelegten Suchwert und liefert im Fall eines Treffers die übrigen Informationen der betreffenden Zelle.



Anwendung: *Memory Management Unit (MMU)* Motorola MC 68851

Zusammenfassung



Translation look aside buffer (TLBs) dienen dem raschen Zugriff auf häufig benutzte Adressumrechnungs-Informationen.

Fehlende Treffer ☞ Umrechnung über den Hauptspeicher.

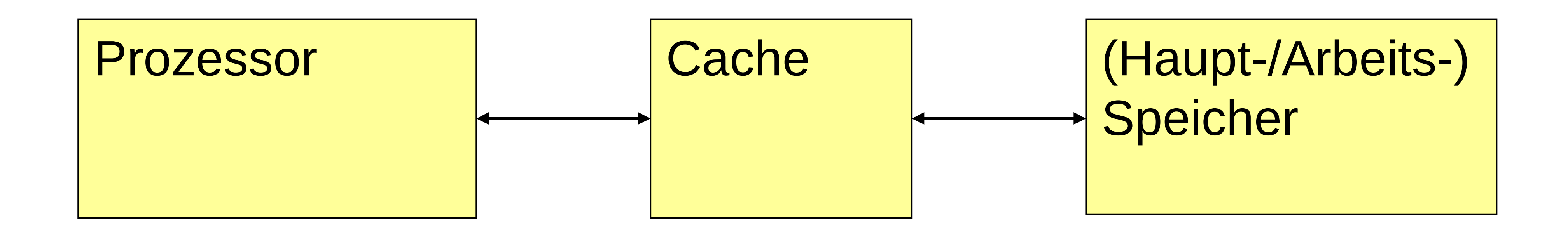
3 Organisationsformen:

1. ***Direct mapping***
2. **Mengen-assoziative Abbildung**
   (set associative mapping), und
3. **Assoziativspeicher *(associative mapping)*.**

TLBs können Teil der *Memory Management Unit* (MMU) sein.

2.4.4 Caches

- ***Cache*** = Speicher, der vor einen größeren, langsamen Speicher geschaltet wird.
- Im weiteren Sinn: Puffer zur Aufnahme häufig benötigter Daten.
- Im engeren Sinn: Puffer zwischen Hauptspeicher und Prozessor.



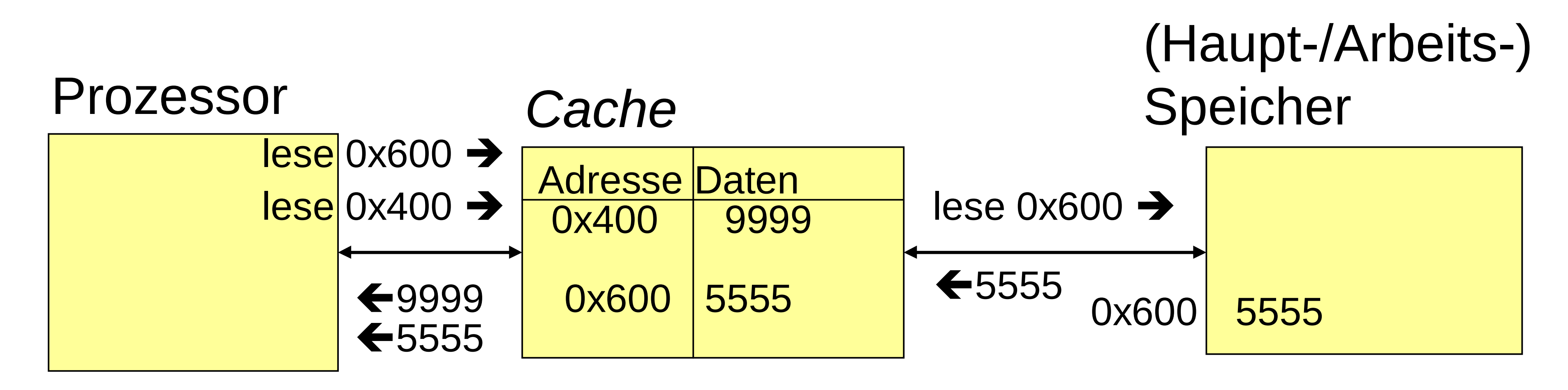
Ursprung***:*** *cacher* (frz.): verstecken („versteckter Speicher").

Ablauf

Organisation von Caches (im engeren Sinn):

Prüfung anhand der (virtuellen oder realen) Adresse, ob benötigte Daten im *Cache* vorhanden sind („Treffer“; *cache hit*).

Falls nicht (*cache miss*): Zugriff auf den (Haupt-) Speicher, Eintrag in den *Cache*.



Prüfung auf *cache hit: Cache*-Zeilen (*cache lines*).

Such-Einheit im Cache: ***Cache*-Zeile** (***cache line***).

Adresse

tag Index

Für *line size* =4.

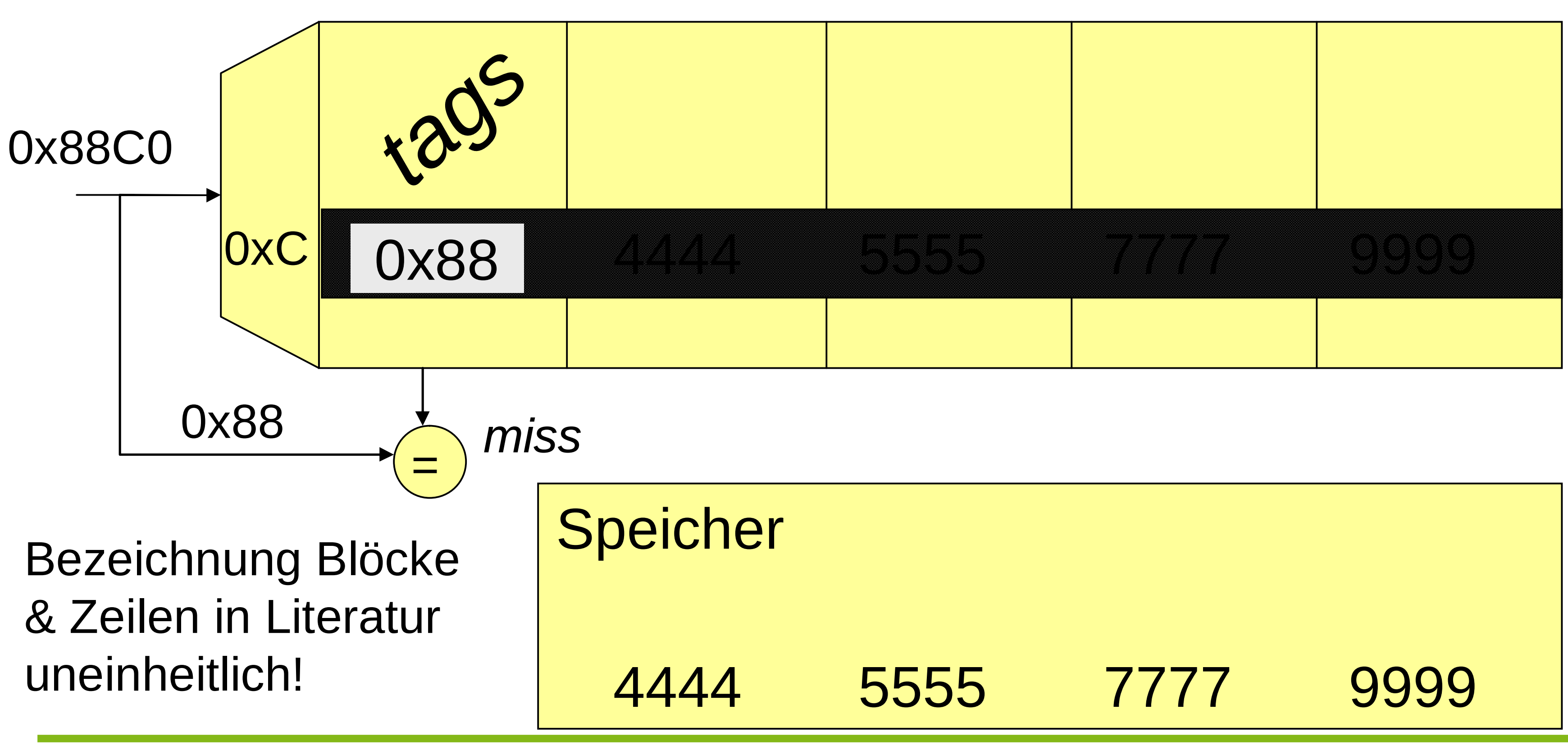
[redacted]

=

Weniger *tag bits*, als wenn man jedem Wort *tag bits* zuordnen würde.

Cache-Blöcke (*cache blocks*) - 1 -

Die **Blockgröße** ist die Anzahl der Worte, die im Fall eines *cache misses* aus dem Speicher nachgeladen werden.
Beispiel: (Blockgröße = *line size):*



Bezeichnung Blöcke & Zeilen in Literatur uneinheitlich!

Cache-Blöcke (*cache blocks*) - 2 -

Wenn *block size* < *line size*, dann sind zusätzliche Gültigkeitsbits erforderlich. Beispiel: Blockgröße = *line size / 2.*

Gültigkeitsbits

Block

tags

0x88C0

0xC

0x88 0 1 7777 9999

0x88

=

miss

Speicher

4444 5555 7777 9999

Cache-Blöcke (*cache blocks*) - 3 -

Wenn *block size* > *line size*, dann werden bei jedem miss mehrere Zeilen nachgeladen.
Stets wird zuerst das gesuchte Wort, dann der Rest des Blocks geladen.
Verbindung Speicher/Cache so entworfen, dass der Speicher durch das zusätzliche Lesen nicht langsamer wird:
Methoden dazu:

1. Schnelles Lesen aufeinanderfolgender Speicherzellen (*nibble mode, block/page mode* der Speicher)
2. *Interleaving* (mehrere Speicher-ICs mit überlappenden Zugriffen)
3. Block-Mode beim Buszugriff, Page-Mode beim Speicherzugriff
4. Fließbandzugriff auf den Speicher (EDO-RAM, SDRAM)
5. breite Speicher, die mehrere Worte parallel übertragen können.

Organisationsformen von Caches

- ***Direct mapping***
  Für *caching* von Befehlen besonders sinnvoll, weil bei Befehlen *aliasing* sehr unwahrscheinlich ist.
- ***Set associative mapping***
  Sehr häufige Organisationsform, mit Set-Größe=2 oder 4, selten 8.
- ***Associative mapping***
  Wegen der Größe eines Caches kommt diese Organisationsform kaum in Frage.

Cache-Simulation in MARS

MARS erlaubt die Cache Simulation.

Achtung:

block size = line size

set size = Assoziativität

Data Cache Simulation Tool, Version 1.2

Simulate and illustrate data cache performance

Cache Organization

Placement Policy	Direct Mapping	Number of blocks	8
Block Replacement Policy	LRU	Cache block size (words)	4
Set size (blocks)	1	Cache size (bytes)	128

Cache Performance

Memory Access Count	3	Cache Block Table
Cache Hit Count	2	(block 0 at top)
Cache Miss Count	1	= empty / = hit / = miss
Cache Hit Rate	67%	

Runtime Log

Enabled

```
  trying block 0 tag 0x00200200 -- HIT
(3) address: 0x10010008 (tag 0x00200200)  block range: 0-0
  trying block 0 tag 0x00200200 -- HIT
```

Tool Control

Disconnect from MIPS | Reset | Close

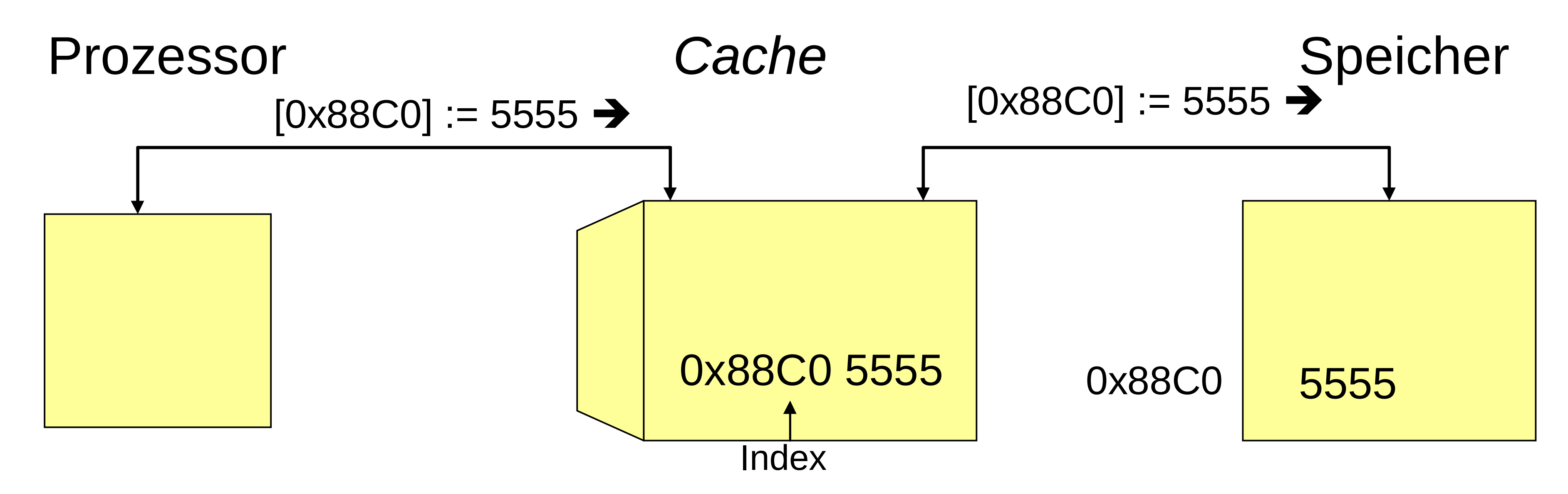
Schreibverfahren



Strategien zum Rückschreiben *Cache* -> (Haupt-) Speicher

1. Write-Through (durchschreiben)

Jeder Schreibvorgang in den *Cache* führt zu einer unmittelbaren Aktualisierung des (Haupt-) Speichers

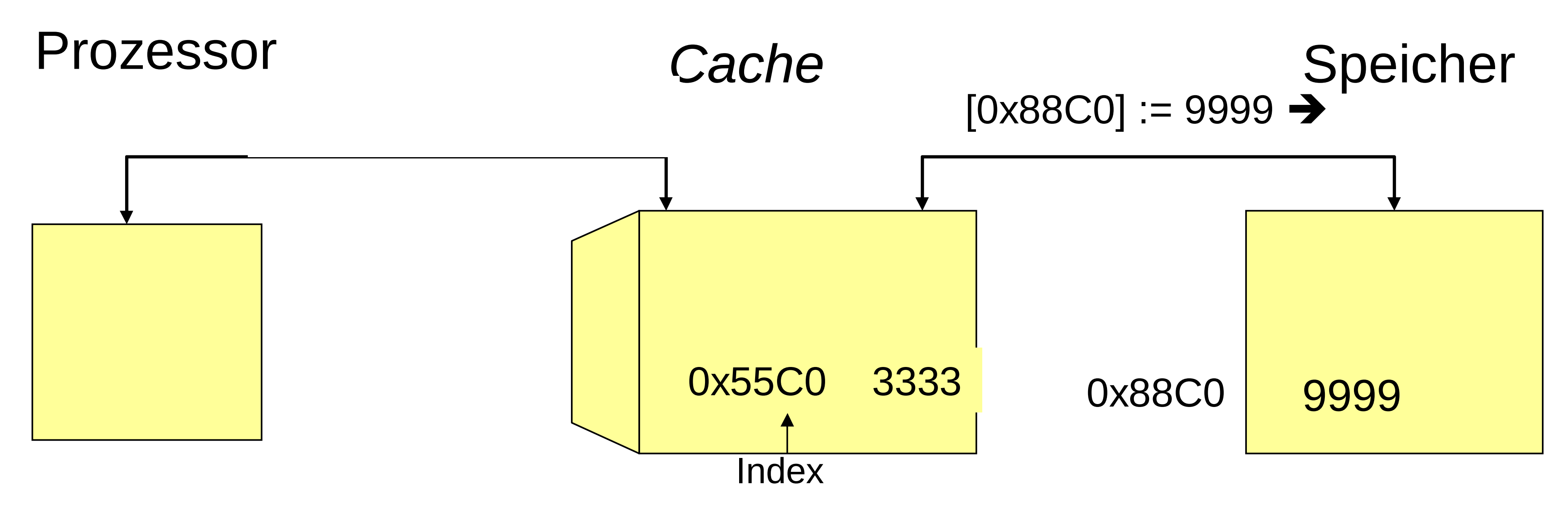


Speicher wird Engpass, es sei denn, der Anteil an Schreiboperationen ist klein oder der (Haupt-) Speicher ist nur wenig langsamer als der *Cache*.

Schreibverfahren



2. ***Copy-Back, conflicting use write back:***
Rückschreiben erfolgt erst, wenn die Cache-Zeile überschrieben wird.



Funktioniert auch bei großen Geschwindigkeitsunterschieden zwischen *Cache* und Speicher. Vorkehrungen erforderlich, damit keine veralteten Werte aus dem Speicher kopiert werden.

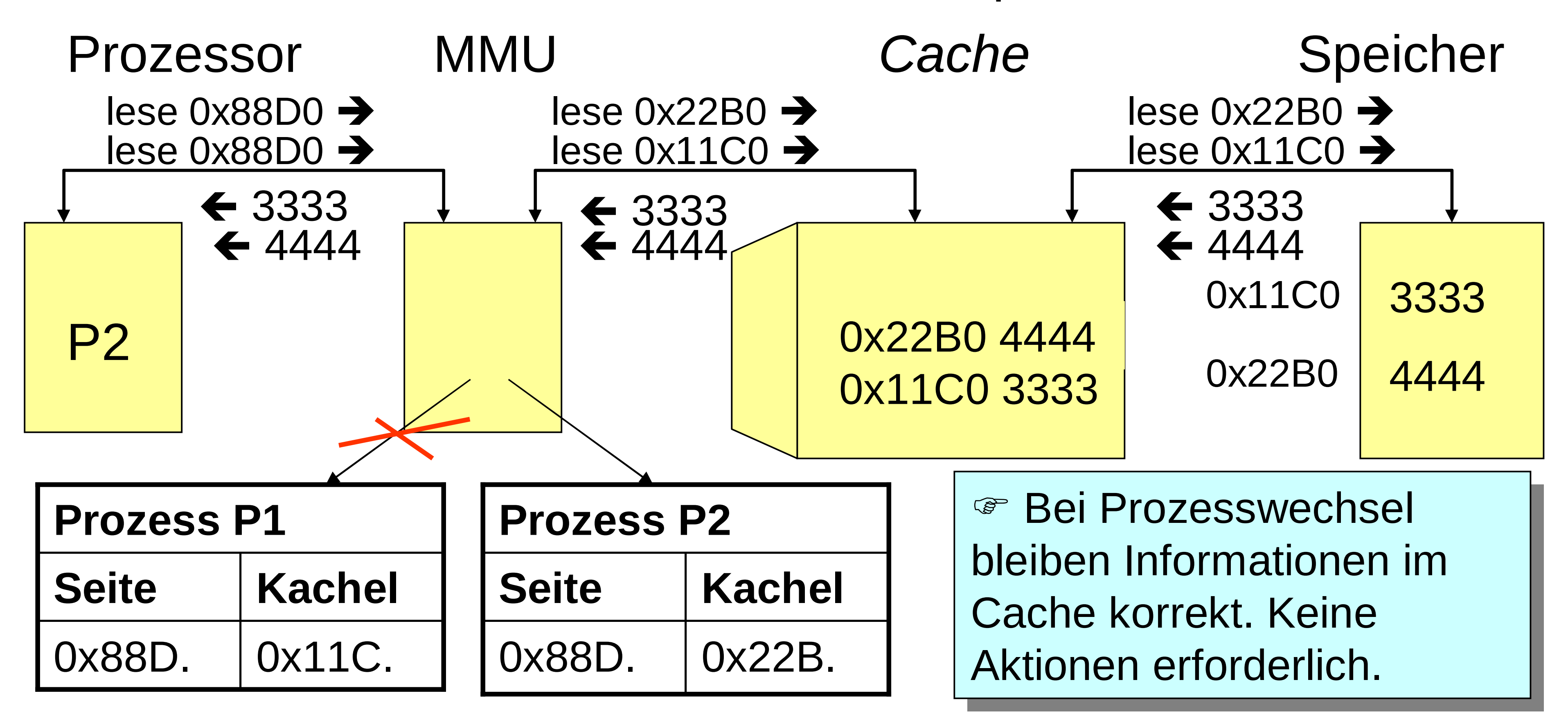
Virtuelle und reale *Caches*

Virtuelle *Caches* (schnell)

Reale *Caches* (langsamer):

Realer *Cache* bei Prozesswechsel (*context switch*)

Zu einer bestimmten Adresse im Cache gehört vor und nach dem Prozesswechsel dieselbe Zelle im Speicher.



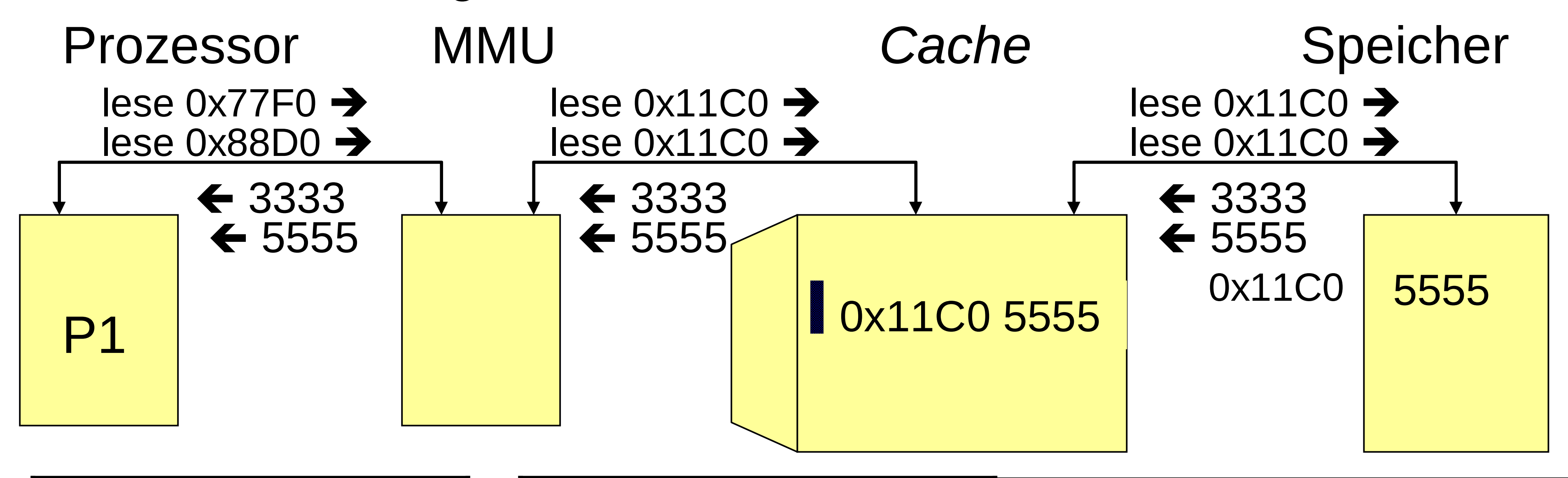
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x22B.

Eintrag 0x11C0 ist nach Rückschalten auf P1 immer noch korrekt.
Einträge überleben evtl. mehrere Prozesswechsel! ☞ Gut für große Caches!

Realer *Cache* bei Seitenfehler
(Seite muss per *demand paging* eingelagert werden)

Zu einer bestimmten Adresse im Cache gehört vor und nach dem Aus- & Einlagern von Seiten eine andere Information.



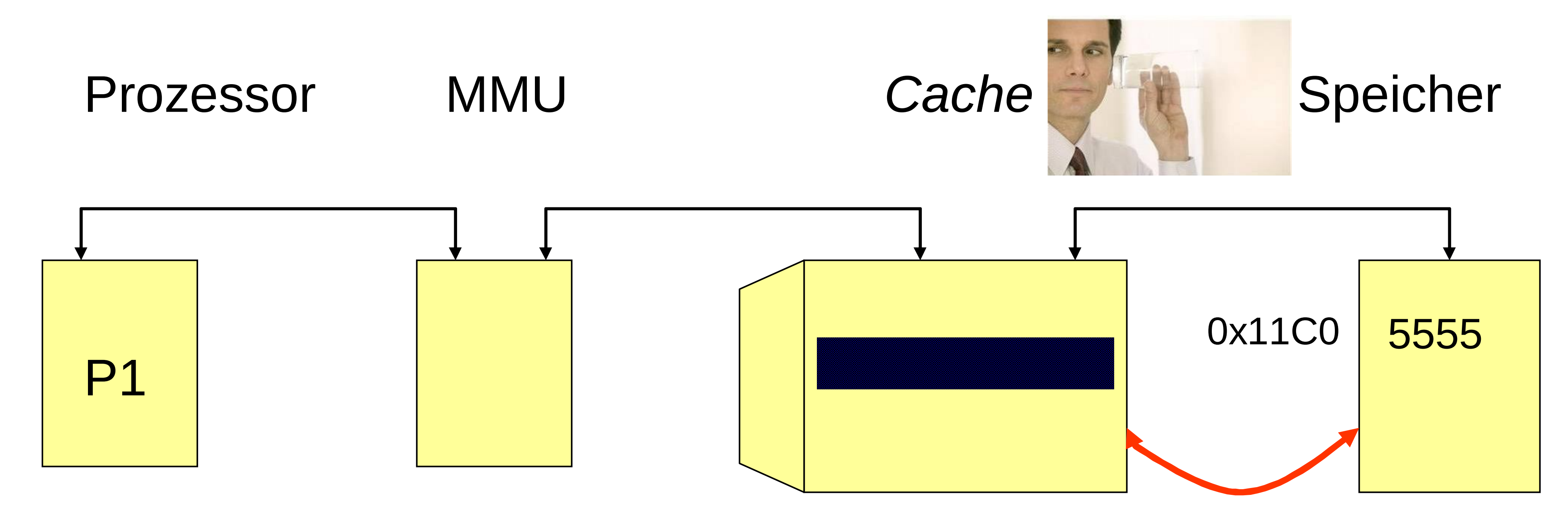
Vor Aus/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

Nach Aus/Einlagern	
Seite	**Kachel**
0x77F.	0x11C.
0x88D.	☞ Platte

☞ Bei Seitenfehlern müssen alle Cache-Zeilen, die Kopien der verdrängten Seite enthalten, ungültig erklärt werden.

Bus snooping (Bus-Lauschen)

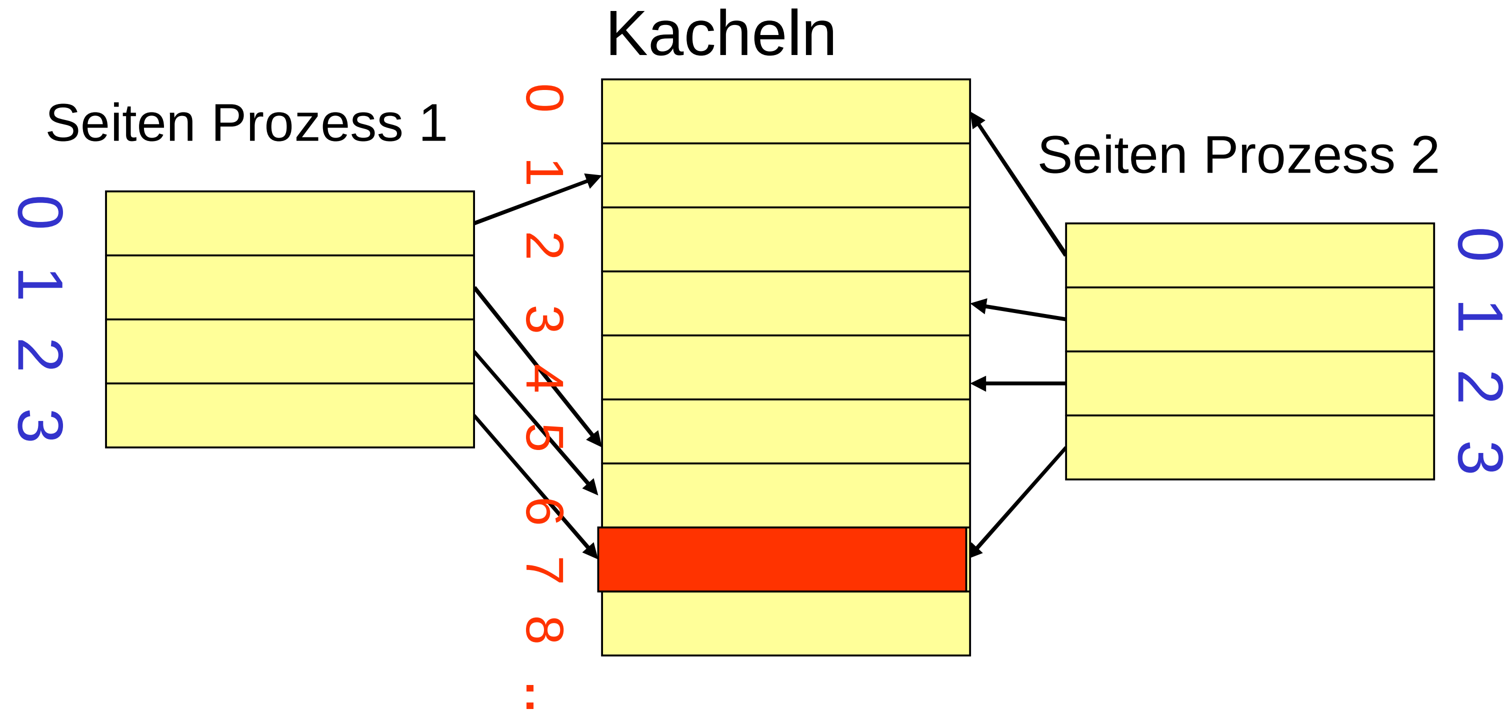
☞ Bei Seitenfehlern müssen alle *Cache*-Zeilen, die Kopien der verdrängten Seite enthalten, ungültig erklärt werden.



Cache „lauscht“, ob ein Schreibvorgang in den Speicher eine Zelle betrifft, von der er eine Kopie hat.
Wenn ja, dann erklärt er diese für ungültig.

Gemeinsam genutzte Seiten (*sharing*)





Für gemeinsame Daten oder gemeinsame Befehle:
Eintrag derselben Kachel in mehrere Seitentabellen.

Realer *Cache* bei gemeinsam genutzter Seite

Annahme *: Beide Prozesse benutzen dieselbe virtuelle Adresse für den gemeinsamen Speicherbereich

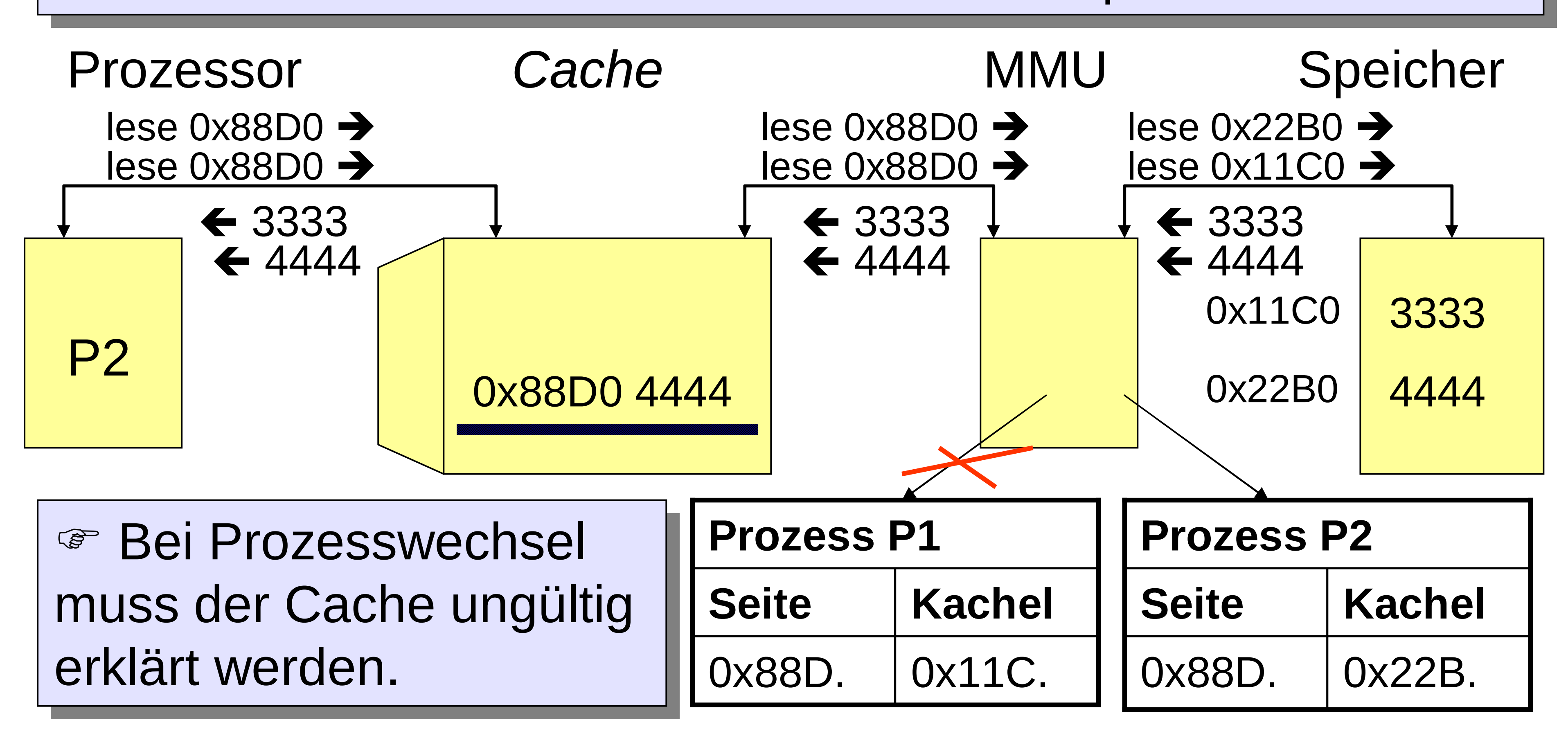
Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x11C.

☞ Nach Prozesswechsel erhält P2 aus dem Cache die richtigen Informationen (unabhängig von *).

Virtueller *Cache* bei Prozesswechsel (*context switch*)

Zu einer bestimmten Adresse im Cache gehört vor und nach dem Prozesswechsel **eine andere** Zelle im Speicher.



Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x22B.

☞ Bei Prozesswechsel muss der Cache ungültig erklärt werden.

Bei jedem Prozesswechsel gehen alle Informationen im Cache verloren.
☞ Schlecht für große Caches!

Virtueller *Cache* bei Seitenfehler

Zu einer bestimmten Adresse im Cache gehört vor und nach dem Seitenfehler dieselbe Information.

☞ Bei Seitenfehlern keine Aktion im *Cache*.

Vor Aus-/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

Nach Aus-/Einlagern	
Seite	**Kachel**
0x77F.	0x11C.

Virtueller *Cache* bei gemeinsam genutzter Seite (*sharing*)

Annahme: Beide Prozesse benutzen dieselbe virtuelle Adresse für den gemeinsamen Speicherbereich

Prozessor | *Cache* | MMU | Speicher

lese 0x88D0 ➔
lese 0x88D0 ➔

🡐 3333
🡐 3333

P2

0x88D0 3333

lese 0x88D0 ➔
lese 0x88D0 ➔

🡐 3333
🡐 3333

lese 0x11C0 ➔
lese 0x11C0 ➔

🡐 3333
🡐 3333

0x11C0

3333

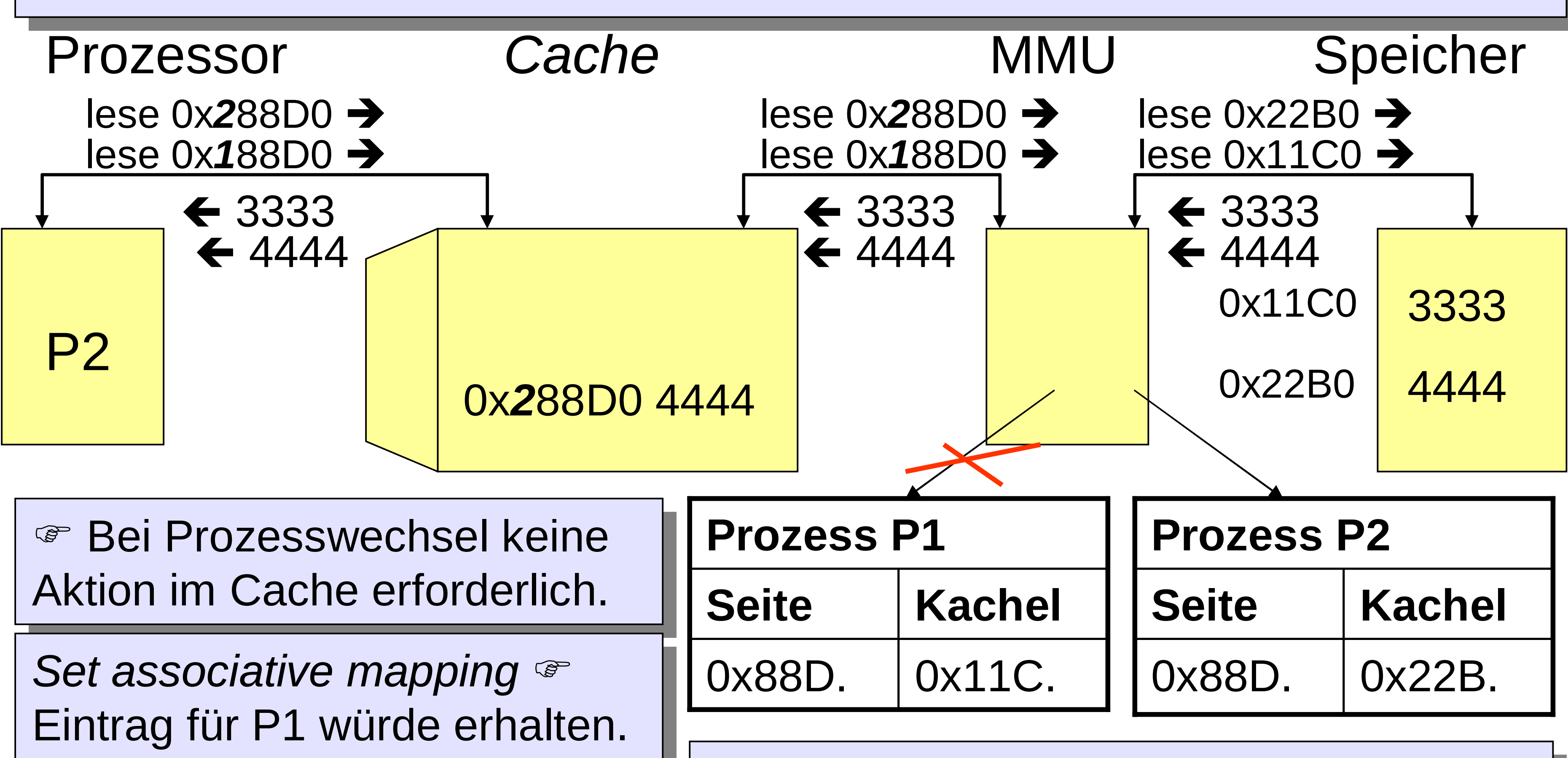
☞ Der zweite Prozess füllt den Cache erneut. Langsam, aber korrekt.

Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x11C.

Virtueller *Cache* mit PIDs bei Prozesswechsel (*context switch*)

Virtuelle Adresse wird um Prozessnummer (*process identifier*, PID) erweitert. Zu einer bestimmten Adresse im Cache gehört vor und nach dem Prozesswechsel wieder dieselbe Zelle.



☞ Bei Prozesswechsel keine Aktion im Cache erforderlich.

Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x22B.

Set associative mapping ☞ Eintrag für P1 würde erhalten.
☞ Gut für große Caches!

TLBs in der MMU könnten PIDs nutzen.

Virtueller *Cache* mit PIDs bei Seitenfehler

Zu einer bestimmten Adresse im Cache gehört vor und nach dem Seitenfehler dieselbe Information.

☞ Bei Seitenfehlern keine Aktion im *Cache*.

Vor Aus-/Einlagern	
Seite	**Kachel**
0x88D.	0x11C.

Nach Aus-/Einlagern	
Seite	**Kachel**
0x77F.	0x11C.

Virtueller *Cache* mit PIDs bei gemeinsam genutzter Seite (*sharing*)

Prozessor | *Cache* | MMU | Speicher

[0x**1**88D0]:=7777 ➔
lese 0x**2**88D0 ➔
lese 0x**1**88D0 ➔

lese 0x**2**88D0 ➔
lese 0x**1**88D0 ➔

lese 0x11C0 ➔
lese 0x11C0 ➔

🡐 3333
🡐 3333

🡐 3333
🡐 3333

🡐 3333
🡐 3333
0x11C0

P2

0x**1**88D0 7777
0x**2**88D0 3333

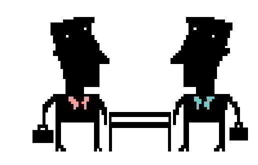
3333

Prozess P1	
Seite	**Kachel**
0x88D.	0x11C.

Prozess P2	
Seite	**Kachel**
0x88D.	0x11C.

Einträge für P1 und P2 können beide im Cache existieren (z.B. bei *set associative mapping)*; sie werden nicht abgeglichen
☞ mögliche **Inkonsistenz** bzw. **Inkohärenz**

Cache-Kohärenz

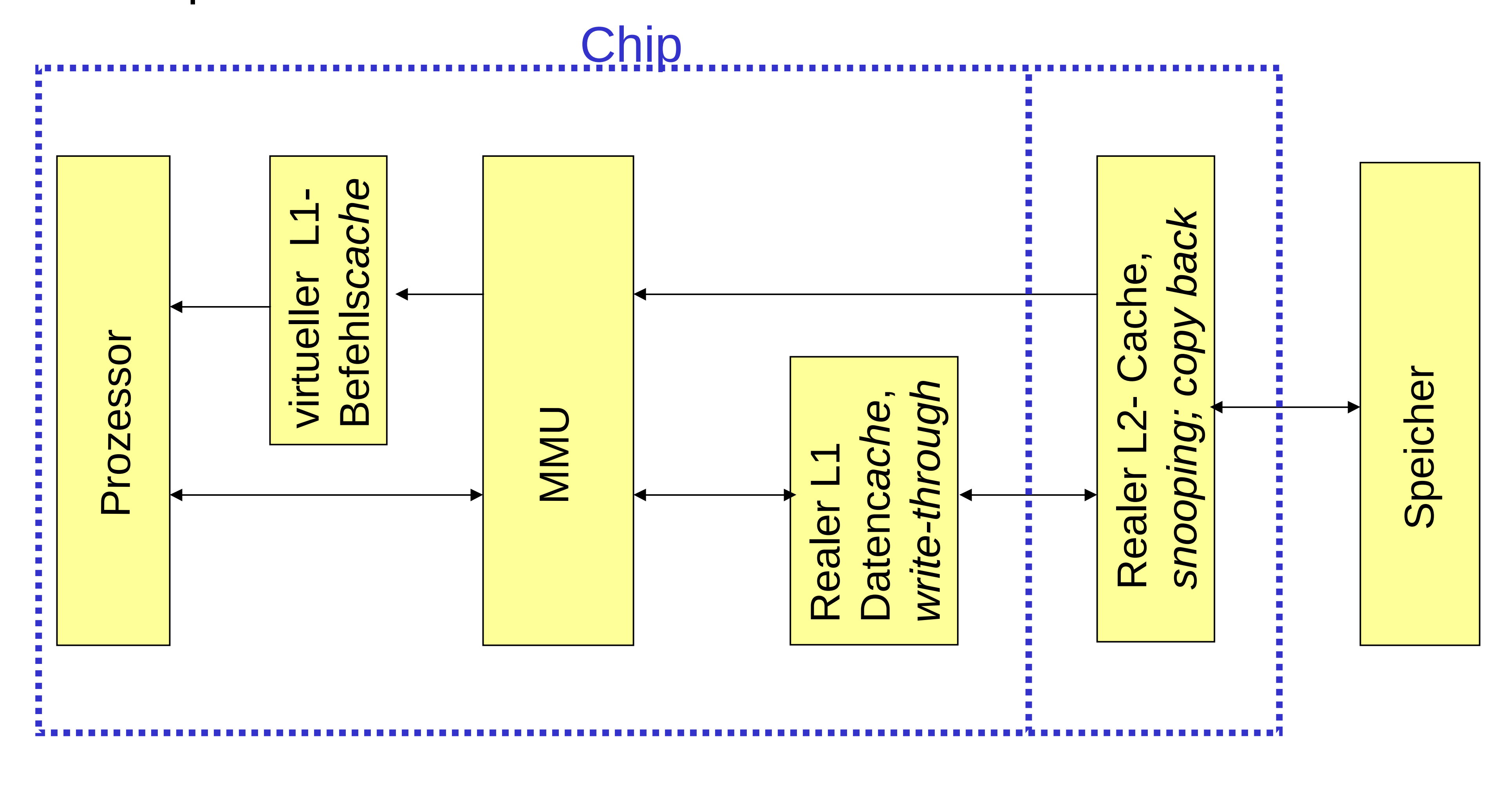


	Realer *Cache*	Virtueller *Cache* mit PIDs	Virtueller *Cache*
Inhalt nach Prozesswechsel	gültig	gültig, falls es ausreichend viele PIDs gibt	ungültig
Inhalt nach Seitenfehler	modifizierte Kachel ungültig	gültig* (verletzt *inclusion property*)	gültig* (verletzt *inclusion property*)
Geschwindigkeit	langsam	schnell	schnell
Automatische Kohärenz bei *Sharing*	ja	nein	ja, aber Code muss neu geladen werden.
primäre Anwendung	große *Caches*	kleine *Caches*; Befehls*caches*, falls dynamisches Überschreiben von Befehlen verboten ist.	

* Potenzielle Schwierigkeiten, wenn Cacheinhalte $\not\subset$ Hauptspeicherinhalte.

Systeme mit mehreren Caches

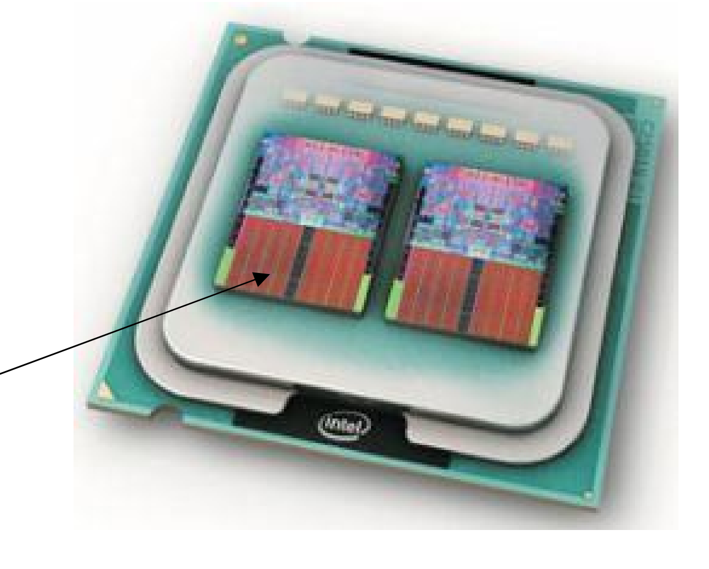
Beispiel:

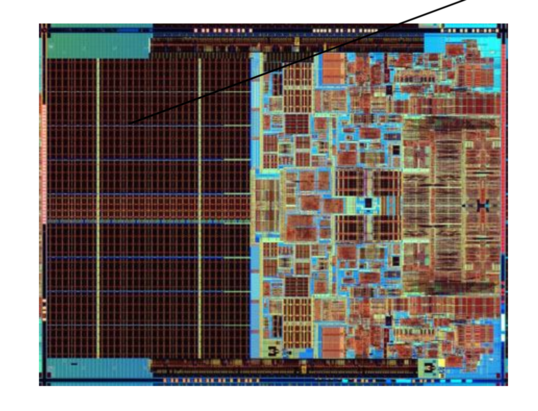


Beispiel: Intel Prozessoren

Beispiel: Intel® Core™ 2 Extreme Quad-Core QX6000:

4 x 32 kB L1 Cache
2 x 4 MB L2 Cache
2 Doppelprozessoren in einem Gehäuse



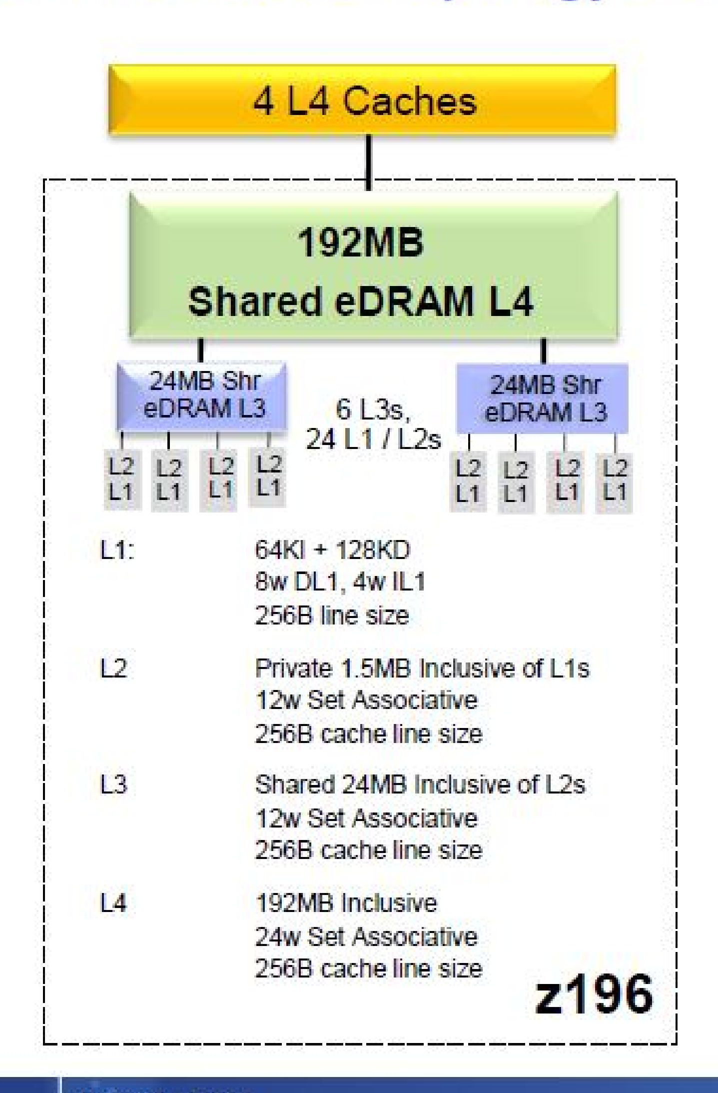


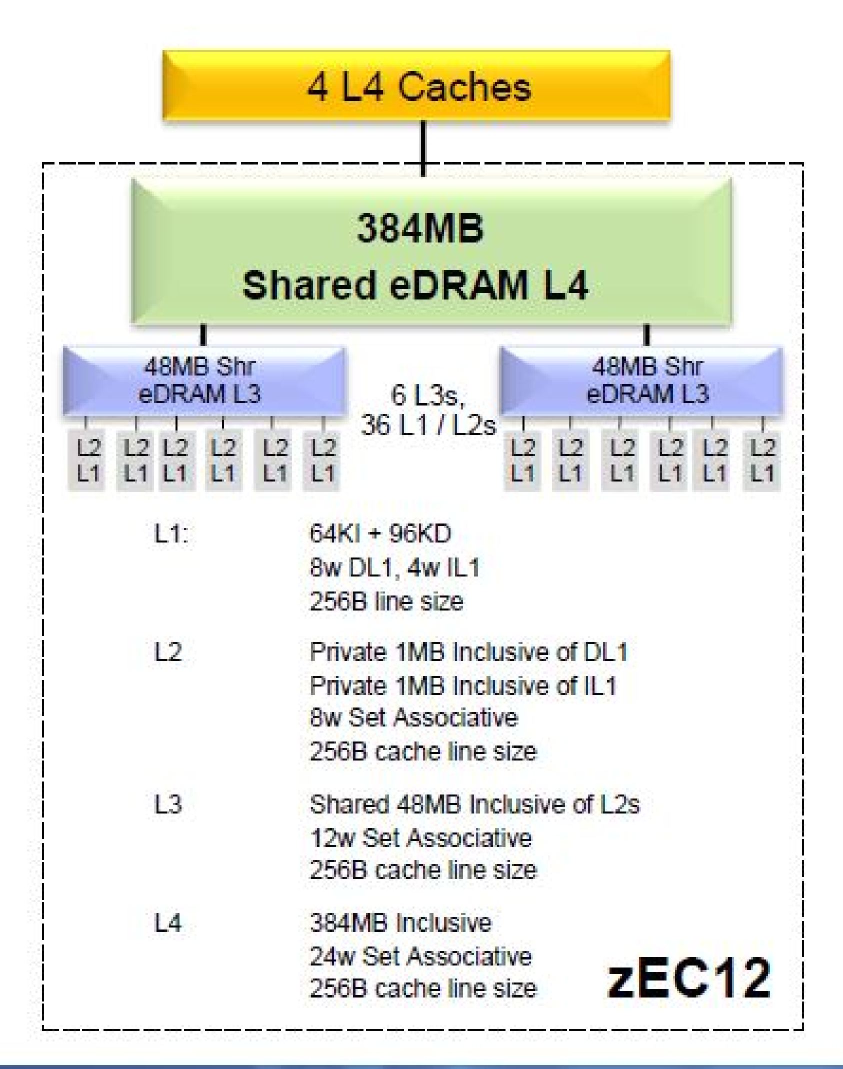
Beispiel: IBM Großrechner

Systems and Technology Group

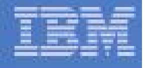
IBM

System z Cache Topology – z196 vs. zEC12 Comparison





Beispiel: IBM Großrechner



zEC12 Book Level Cache Hierarchy

Cache Comparison	
z196	zEC12
L1 – 64K/128K	L1 – 64K/96K
L2 – 1.5 M	L2 – 1 M/1 M
L3 – 24 M	L3 – 48 M
L4 – 192 M	L4– 384 M

Austauschverfahren



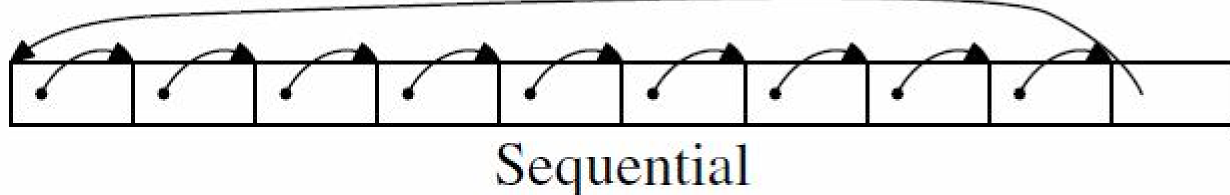
Überschreiben von Einträgen in TLB, Caches, Kacheln:

1. **Random- bzw. Zufallsverfahren**
   Ein zufällig ausgewählter Eintrag wird überschrieben.
2. **NRU, *not recently used***
   Einteilung von Einträgen in 4 Klassen:
   a. **Nicht benutzte, nicht modifizierte Einträge**
   b. **Nicht benutzte, modifizierte Seiten**
      Benutzt-Kennzeichnung wird nach einiger Zeit zurückgesetzt.
   c. **Benutzte, nicht modifizierte Seiten**
   d. **Benutzte, modifizierte Seiten**

   Überschrieben wird ein zufällig ausgewählter Eintrag der niedrigsten Klasse, die nicht-leer ist.
3. **LRU=*least recently used***
   Überschreiben des Eintrags, der am längsten nicht benutzt wurde.

Wie wirkt sich der Cache auf die Laufzeit von Programmen aus?

- Untersuchung der Laufzeit beim Durchlauf durch eine im Array abgelegte verkettete Liste mit Einträgen von NPAD*8 Byte



- Pentium P4
- 16 kB L1 Daten-Cache mit 4 Zyklen/Zugriff
- 1 MB L2 Cache mit 14 Zyklen/Zugriff
- Hauptspeicher mit 200 Zyklen/Zugriff

U. Drepper: *What every programmer should know about memory**, 2007, http://www.akkadia.org/drepper/cpumemory.pdf; Dank an Prof. Teubner (LS6) für Hinweis auf diese Quelle
* In Anlehnung an das Papier „David Goldberg, *What every programmer should know about floating point arithmetic, ACM Computing Surveys*, 1991 (auch für diesen Kurs benutzt).

Zyklen/Zugriff als Funktion der Listengröße

L1d L2-Treffer L2-Miss*

Cycles/List Element

10
9
8
7
6
5
4
3
2
1
0

2^{10} 2^{13} 2^{16} 2^{19} 2^{22} 2^{25} 2^{28}

Working Set Size (Bytes)

Figure 3.10: Sequential Read Access, NPAD=0

L1d L2-Treffer L2-Miss§

Cycles/List Element

350
300
250
200
150
100
50
0

2^{10} 2^{13} 2^{16} 2^{19} 2^{22} 2^{25} 2^{28}

Working Set Size (Bytes)

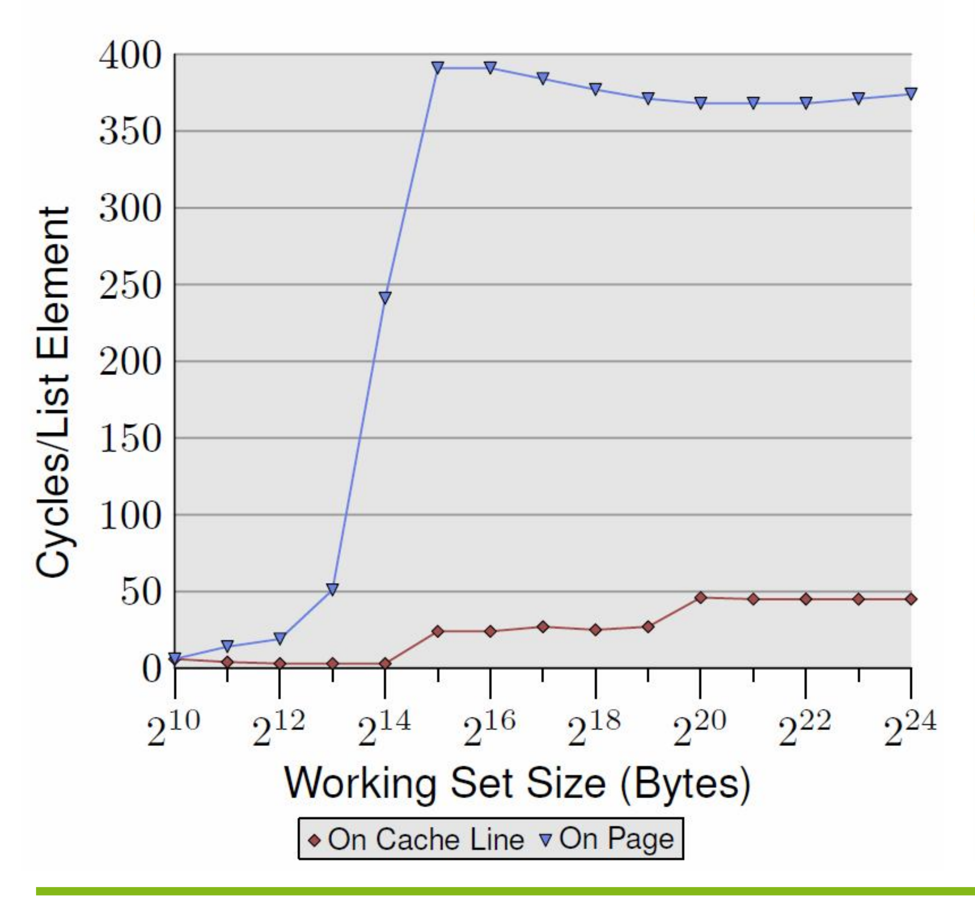
◆ NPAD=0 ▼ NPAD=7 ▲ NPAD=15 ▶ NPAD=31

* Funktionierendes Prefetching

§ Prefetching schlägt fehl

Einfluss von TLB-Misses und größeren Caches

Elemente auf verschiedenen Seiten; Anwachsen der Laufzeit, wenn TLB-Größe überschritten wird



Größere Caches verschieben die Stufen

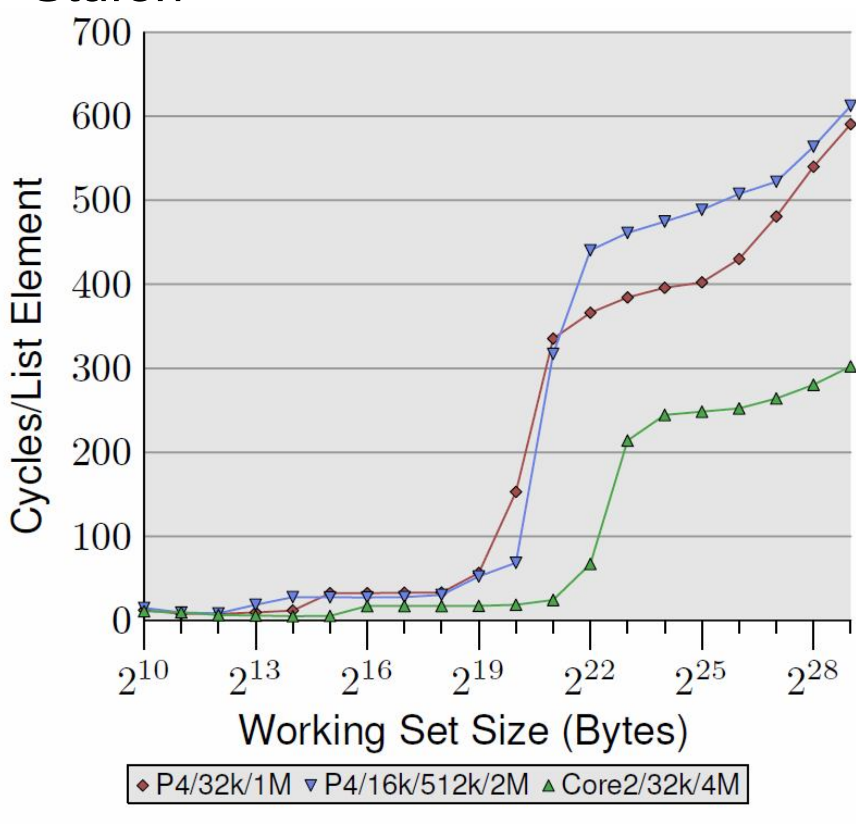


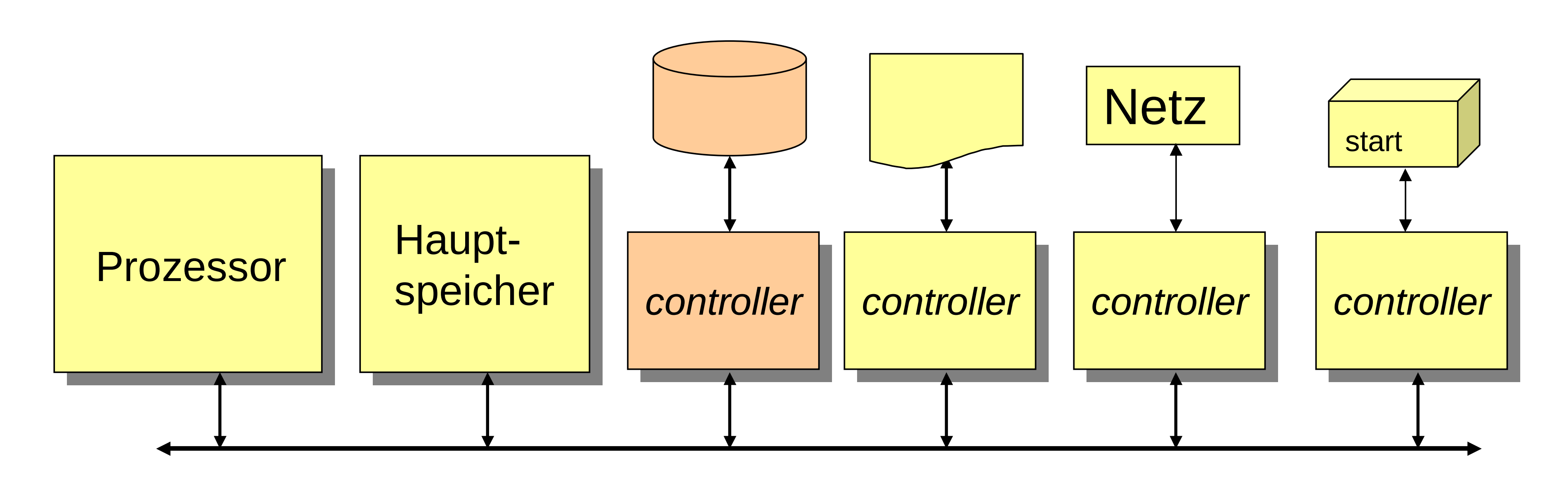
Figure 3.14: Advantage of Larger L2/L3 Caches

Zusammenfassung



- Cache-Zeilen: Einheit der Gültigkeitsprüfung
- Cache-Blöcke: Einheit des Nachladens
- Organisationsformen:
  - *direct mapping*
  - *set associative mapping*
- Unterscheidung zwischen virtuellen und realen Caches.
- Moderne Systeme besitzen mehrere Cache-Ebenen

Massenspeicher



- Plattenspeicher
  - Disc-Arrays
- Flash-Speicher

- CD-ROM
- DVD
- Bandlaufwerke
- Weitere

Mögliche Stufen der Speicherhierarchie
und derzeit eingesetzte Technologien

Tertiärspeicher (Optische Speicher, Bänder)

↕

Sekundärspeicher (Platten, Flash)

↕

Plattencaches (DRAM, Flash)

↕

Haupt- bzw. Primärspeicher (DRAM)

↕

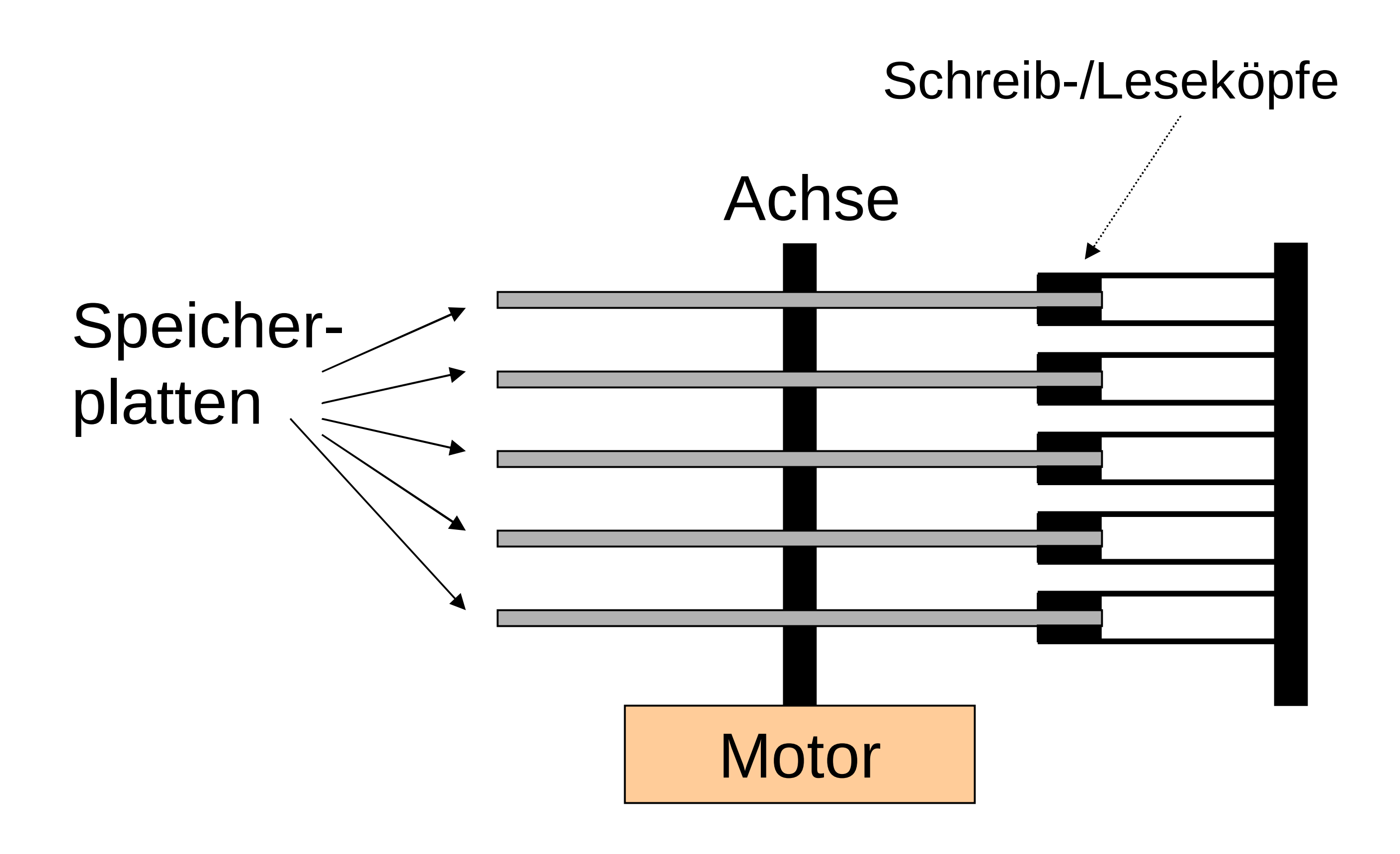
Caches (SRAM) &TLBs

↕

Register (SRAM)

persistent

Schematischer Aufbau eines Plattenlaufwerks



Einteilung der Platten in Sektoren, Spuren und Zylinder

Cylinder
Platters
Tracks
Platter
Sectors
Track

[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

Daten einiger Plattenlaufwerke (2011)

	Seagate Barracuda XT.	Seagate Momentus Thin	Toshiba MK1214GAH
Durchmesser ["]	3,5	2,5”	1,8”
Kapazität [GB]	2000	320	320
Umdrehungen[RPM]	7200	7200	5400
Mittl. Latenz [ms]	4.17 ms	4,17	5,56
Average seek [ms]	8,5 read/9,5 write	11	15
Übertragungsrate	Max. 600 MB/s, 138 MB/s dauerhaft	300 MB/s	300 MB/s
Leistung [W] *passiv/aktiv*	6,39/9,23	0,66/1,6	0,40/1,3
Puffer [MB]	64	16	16
Gb/sq.in	347	425	?
Schock-Toleranz [Betrieb/außer Betrieb]	86 G/ 300 G	350 G/1000 G	500 G/1500 G
Interface	SATA 6 Gb/s	SATA, 3 Gb/s	SATA, 3 Gb/s

Abhängigkeit der Suchzeit von der Spurdifferenz

Access time (ms)

30
25
20
15
10
5
0

0 250 500 750 1000 1250 1500 1750 2000 2250 2500

Seek distance

Naive seek formula

New seek formula

berücksichtigt Beschleunigung

$$\text{"Naive formula": Accesstime(distance)} = Time_{\min} + \frac{\text{distance}}{\text{distance}_{ave}} \times \left(Time_{ave} - Time_{\min}\right)$$

$$\text{"New formula": Access time(distance)} = a \times \sqrt{\text{distance} - 1} + b \times \left(\text{distance} - 1\right) + c$$

[Hennessy/Patterson, *Computer Architecture*, 3. Aufl.]

Verteilung der Spurdifferenzen

Seek distance	Percentage of seeks (UNIX time-sharing workload)
195	3%
180	3%
165	2%
150	3%
135	2%
120	3%
105	3%
90	1%
75	3%
60	3%
45	4%
30	8%
15	23%
0	24%

Seek distance	Percentage of seeks (business workload)
208	0%
192	0%
176	0%
160	0%
144	3%
128	1%
112	1%
96	3%
80	1%
64	1%
48	1%
32	3%
16	11%
0	61%

X-axis scale in both charts: 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%

Interfaces

- IDE/ATA (*Integrated Drive Electronics*):
- SATA (*Serial ATA*):
- SCSI (*Small Computer System Interface*):
- SAS (*serial attached SCSI*): SCSI-Protokoll über serielle, SATA-kompatible Leitung
- *Fibre Channel*: Hochleistungsanschluss für den Server-markt
- USB/FireWire ☞ siehe nächstes Kapitel

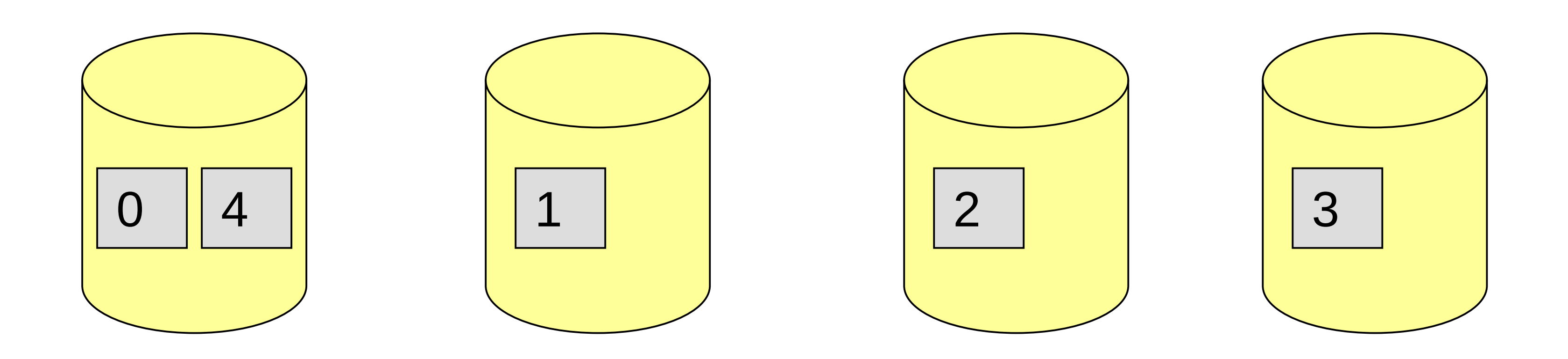
***Redundant arrays of inexpensive discs* (RAID)**

- Verbesserung der E/A-Geschwindigkeit durch paralleles Lesen/Schreiben auf mehrere Platten;
- Einsatz von Redundanz, damit dadurch nicht die Zuverlässigkeit leidet.

☞ *Disc arrays, drive arrays*

RAID 0 (*Striping*)

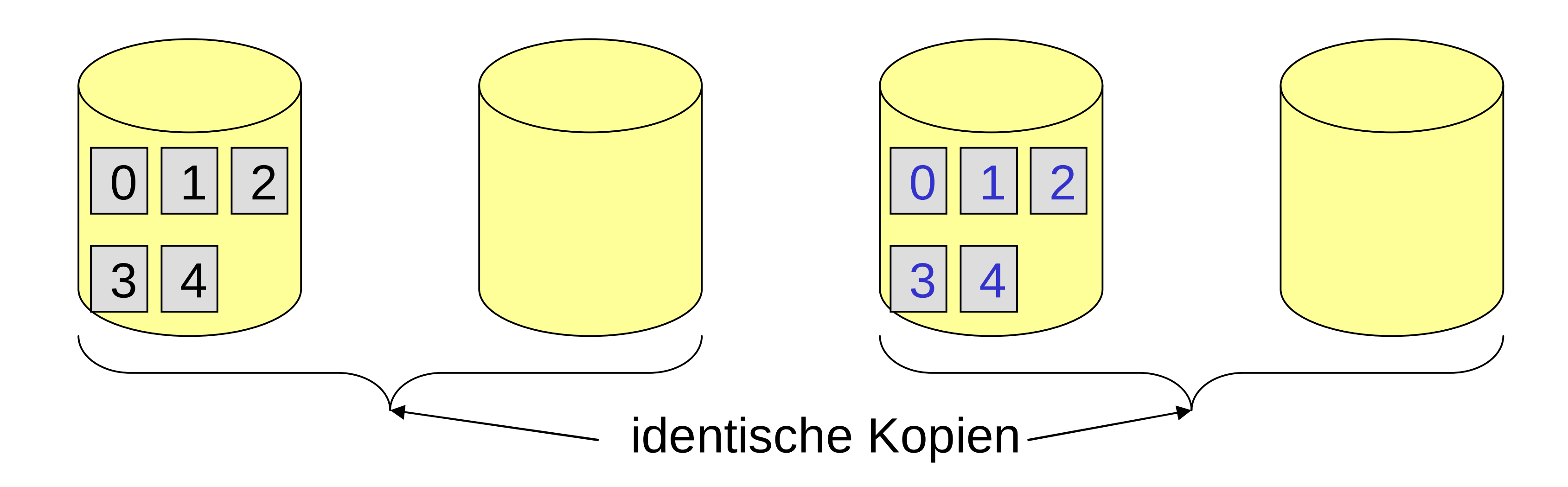
Dateien werden unterteilt in so genannte ***Stripes***.
Striping-Parameter = Länge eines *Stripes.*
Diese *Stripes* werden auf den Platten verteilt.



- Keine Redundanz,
- nur Verbesserung der Übertragungsrate

RAID 1 (*Mirroring*)

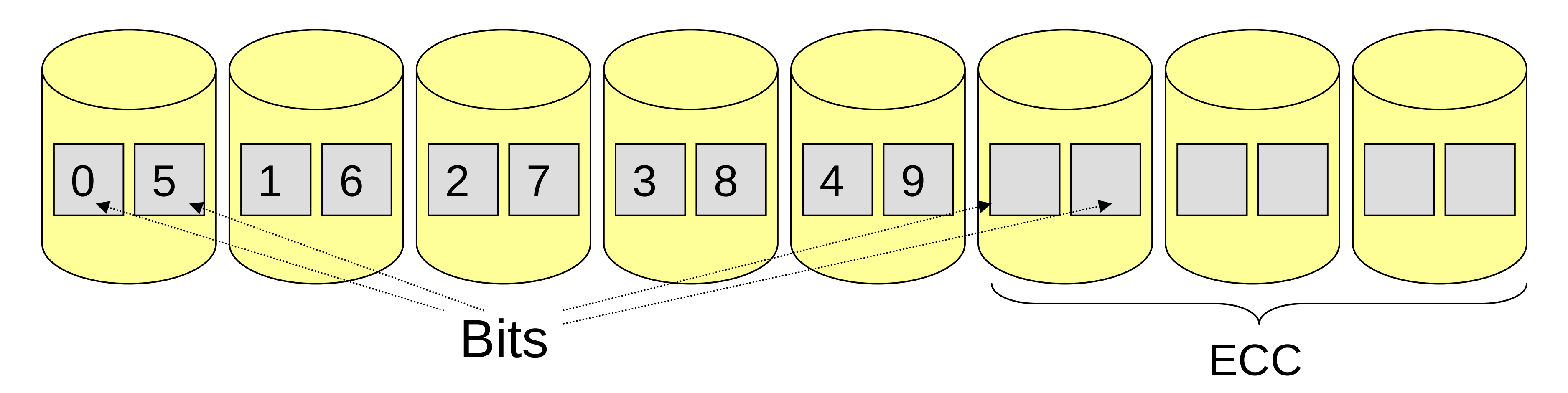
Dieselben Informationen werden auf zwei Platten geschrieben (gespiegelte Platten, *mirrored discs*)



- Erhöhte Fehlersicherheit,
- Stark erhöhter Aufwand (x 2),
- Es werden eigentlich nur 2 Plattenlaufwerke genutzt.

RAID 2 (ECC)

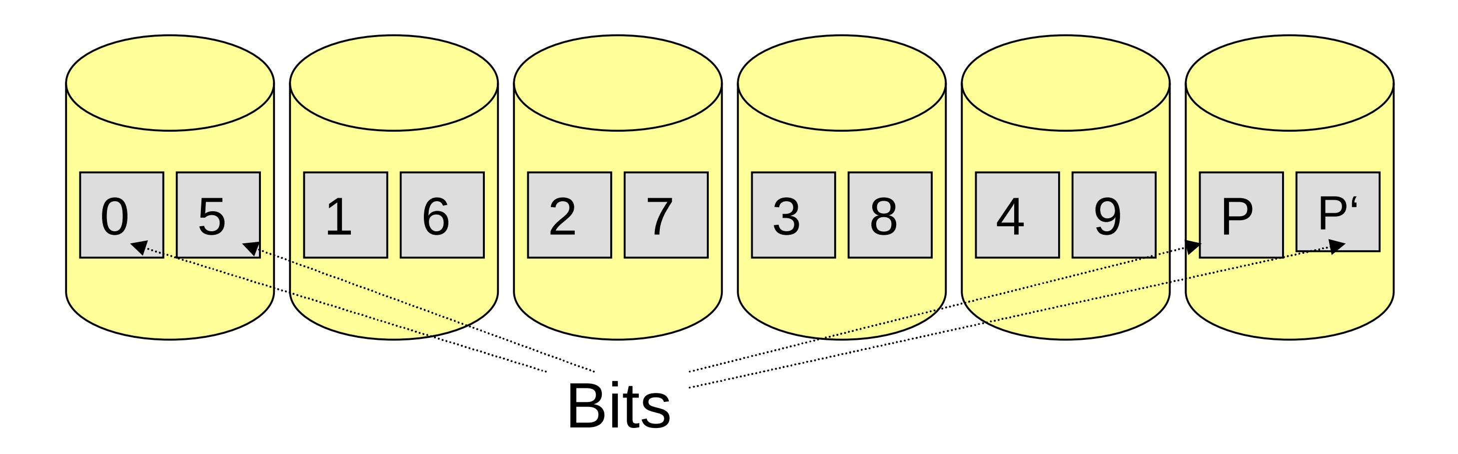
Zusätzlich werden Prüfbits auf spezielle Platten geschrieben, Verwendung von fehlerkorrigierenden Codes (*error correcting code*, ECC).
Die **Bits** werden auf mehreren Platten verteilt.



- Geringere Redundanz als RAID 1, aber wegen Prüfbit-Erzeugung und Last auf Prüfbitplatten langsamer.
- Erfordert spezielle Platten, kommerziell nicht verfügbar.

RAID 3 (*dedicated parity*)

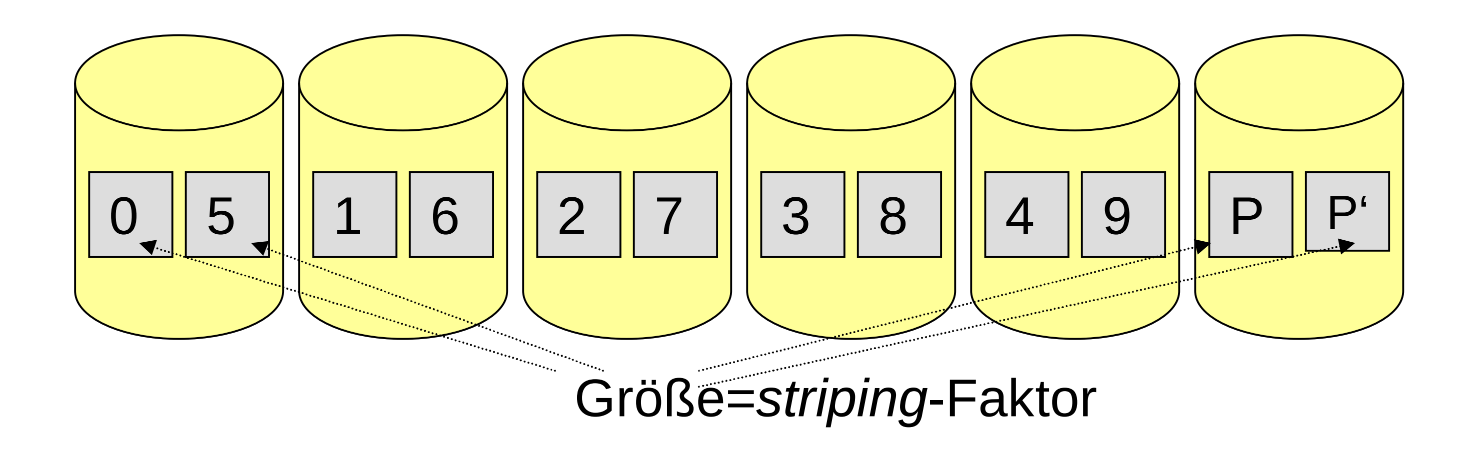
Es nur ein einzelnes Paritäts-*Stripe* auf einer *Parity*-Platte abgelegt.
Häufig nur der Fall *Stripe*-Parameter=1 Bit betrachtet.
Die ***Stripes*** eines Datenblocks werden auf Platten verteilt.



Selbst für kleinste zu schreibende/lesende Blöcke alle Platten aktiv. Keine gute *Performance.*

RAID 4

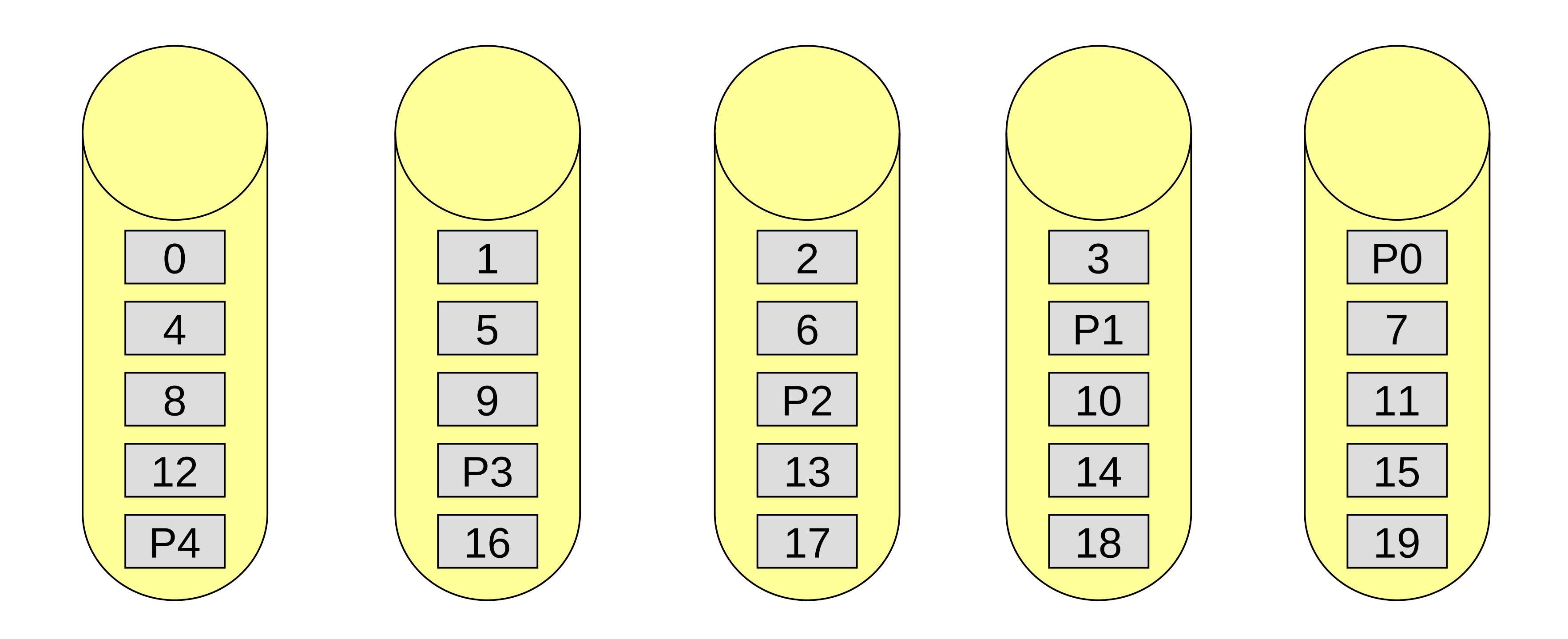
Wie RAID-3, jedoch mit einem *Striping*-Faktor von einem Block und mehr.



Besserer wahlfreier Zugriff als RAID 3.
Paritätsplatte bleibt ein Engpass.

RAID 5 (*distributed parity*)

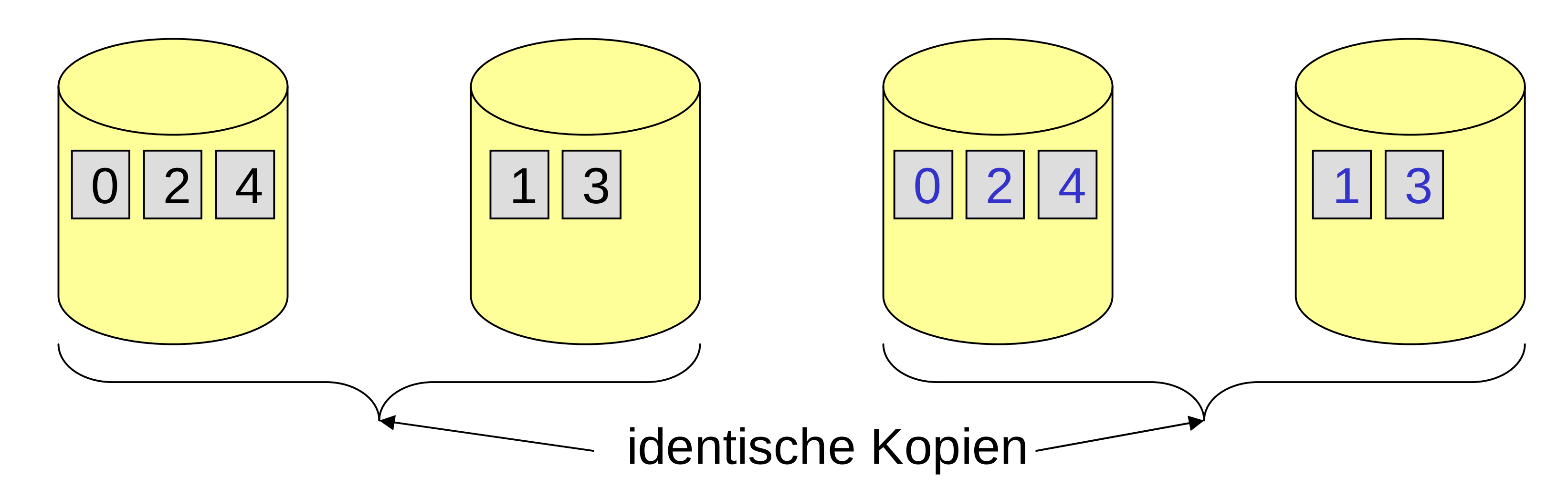
Paritätsinformation wird über verschiedene Platten verteilt



Paritätsplatte nicht weiter ein Engpass

RAID 0+1, RAID 01 (*mirrored stripes*)

Gespiegelte Platten (RAID 1), mit verteilten Stripes (RAID 0) ☞ *mirrored stripes*.



- Lesegeschwindigkeit erhöht,
- erhöhte Fehlersicherheit,
- stark erhöhter Aufwand.

RAID 1+0, RAID 10 (*striped mirrors*)

Striped mirrors: *striping* von gespiegelten Platten

- Lesegeschwindigkeit erhöht,
- erhöhte Fehlersicherheit,
- stark erhöhter Aufwand.

Informationen zu neuen Varianten:
http://www.pcguide.com/ref/hdd/perf/raid/levels/multLevel01-c.html

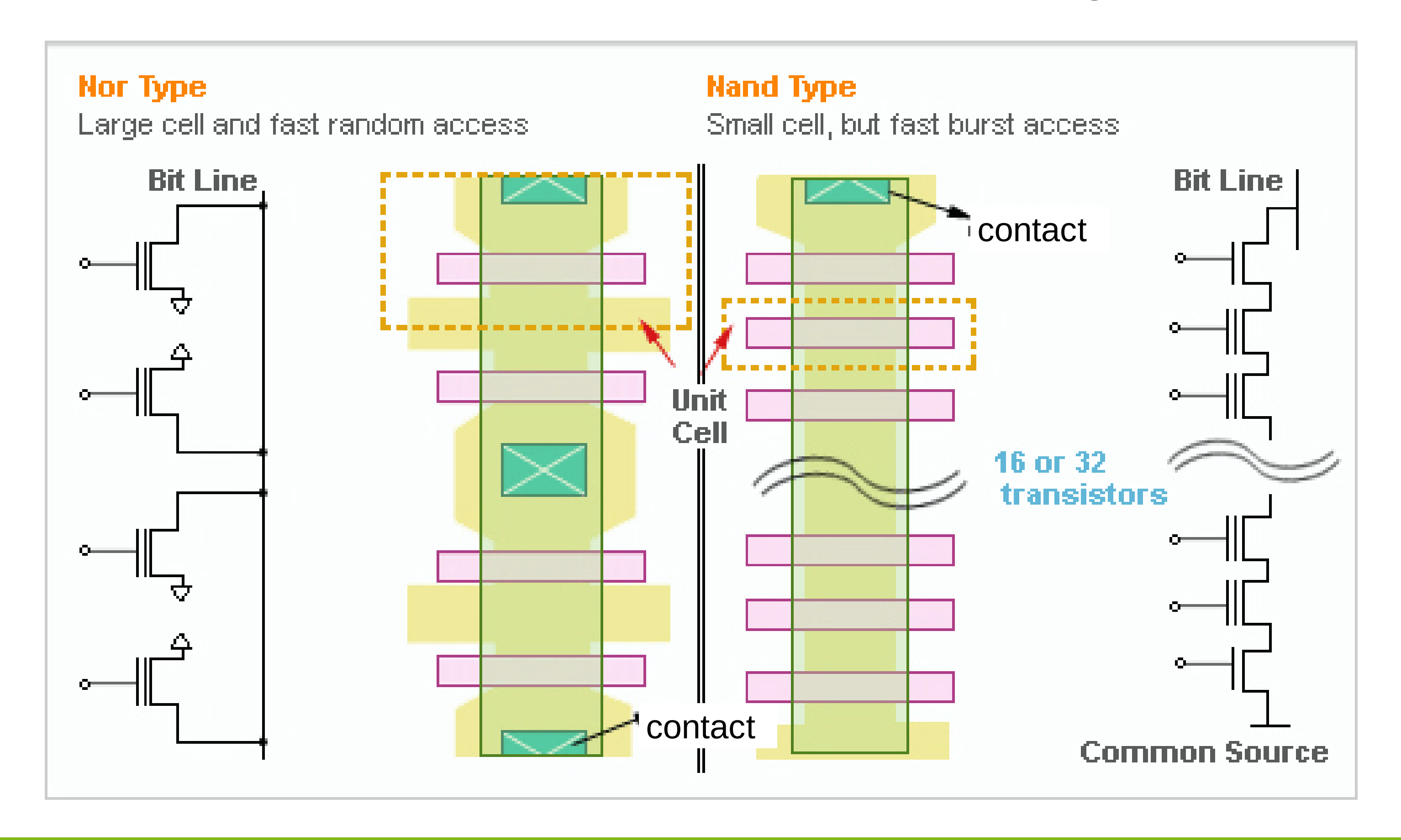
Übersicht

RAID Variante	Technik	Fehlertoleranz	Daten-Platten	Prüfplatten
0	*non-redundant*	0	8	0
1	*mirrored*	1	8	8
0+1	Komb. v. 0 & 1	1	8	8
2	*memory-style ECC*	1	8	4
3	*bit-interleaved parity*	1	8	1
4	*block-interleaved parity*	1	8	1
5	*dto, distributed parity*	1	8	1
6	*P+Q redundancy*	2	8	2

bevorzugt

NOR- und NAND-Flash

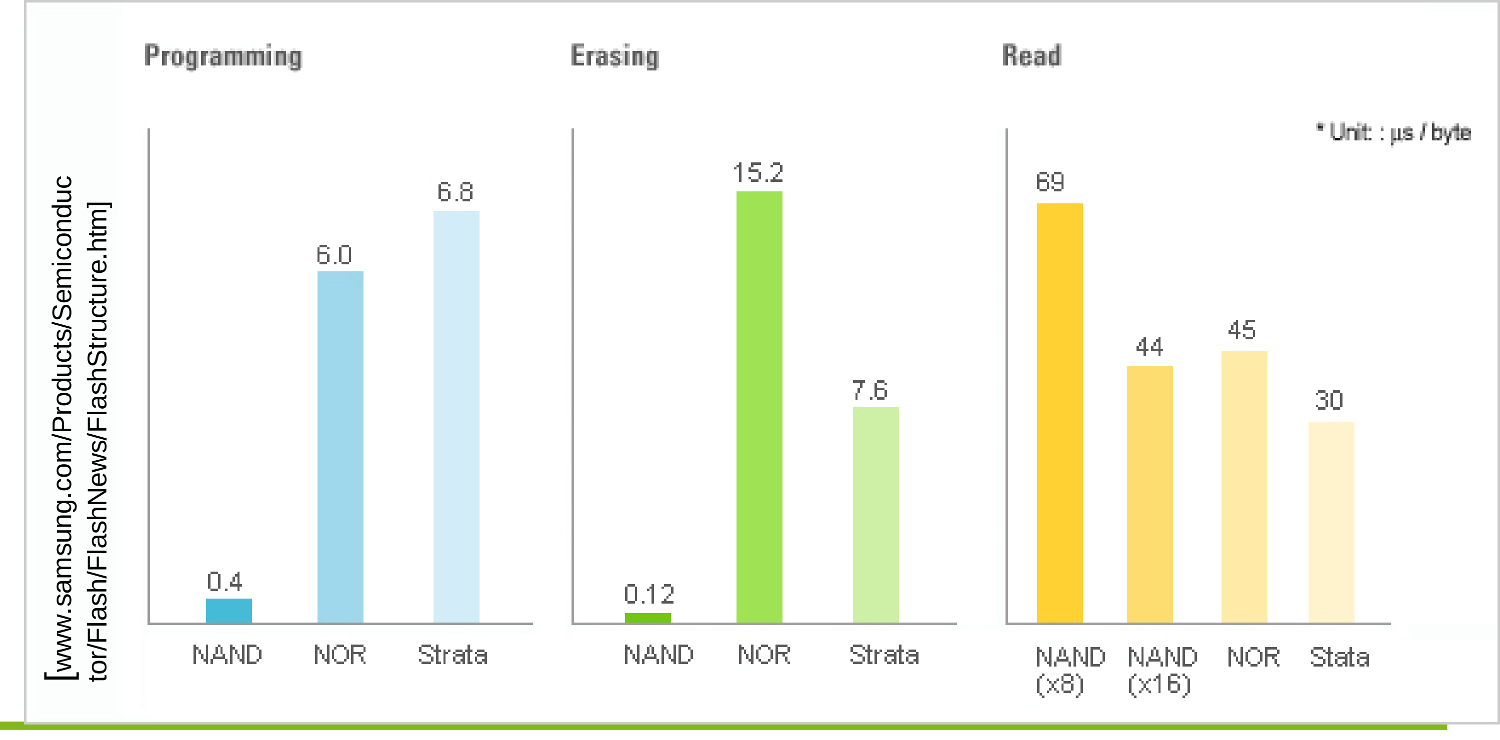
NOR: 1 Transistor zwischen Bitleitung und Masse
NAND: >1 Transistor zwischen Bitleitung und Masse



was at [www.samsung.com/Products/ Semicon-ductor/ Flash/FlashNews/ FlashStructure. htm] (2007)

Eigenschaften von NOR- und NAND-Flash-Speichern

Type/Eigenschaft	NOR	NAND
Wahlfreier Zugriff	Ja ☺	Nein ☹
Block löschen	Langsam ☹	Schnell ☺
Zellgröße	Groß ☹	Klein ☺
Zuverlässigkeit	Größer ☺	Kleiner ☹
Direktes Ausführen	Ja ☺	Nein ☹
Anwendungen	Codespeicherung, *boot flash, set top box*	Datenspeicher, USB *Sticks*, Speicherkarten



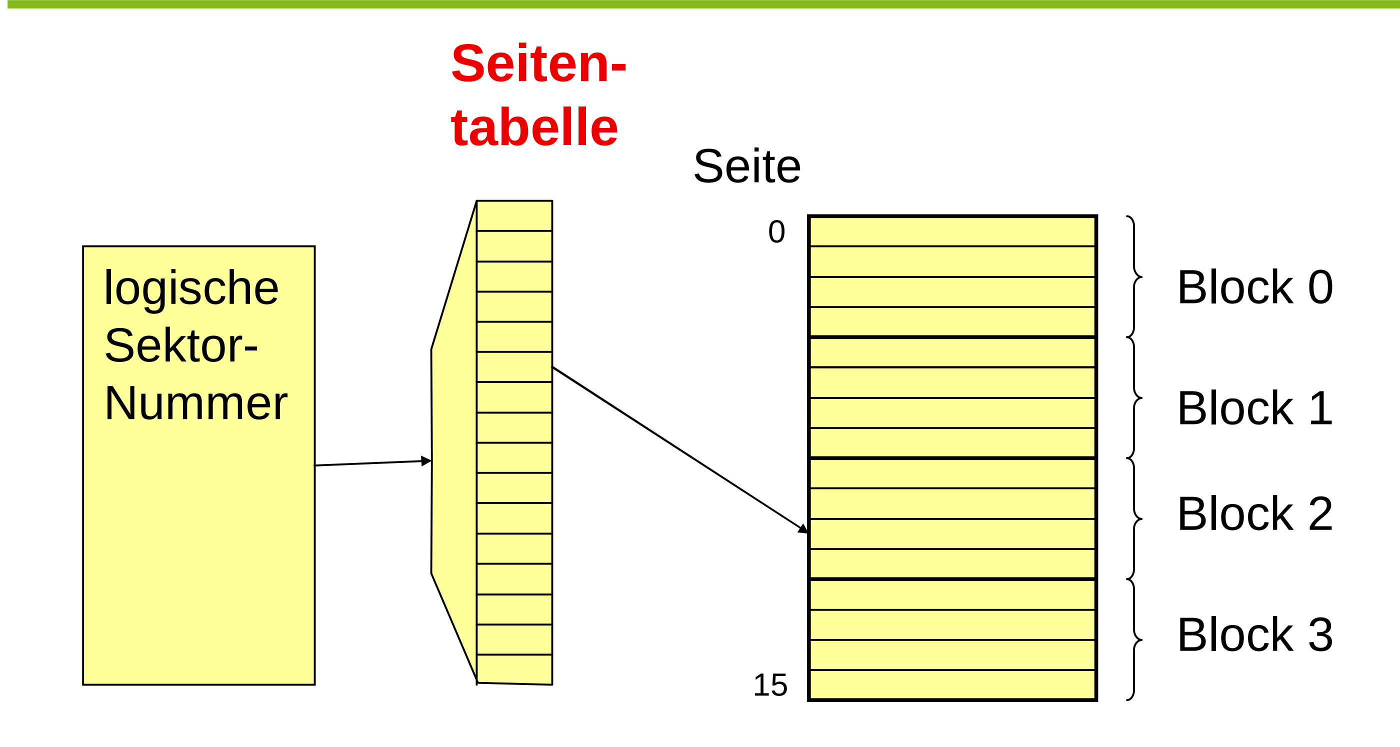
Charakteristische Eigenschaften von NAND Flash Speicher

Speicher aufgeteilt in Blöcke (typ. 16-256 KB),
Blöcke unterteilt in Seiten (typ. 0.5-5 KB).
Schreib-/Lesevorgänge jeweils auf Seiten

	1 Bit/Zelle (SLC)	>1 Bit/Zelle (MLC)
Lesen (Seite)	25 µs	≫ 25 µs
Schreiben (Seite)	300 µs	≫ 300 µs
Löschen (Block)	2 ms	1.5 ms

J. Lee, S. Kim, H. Kwin, C. Hyun, S, Ahn, J. Choi, D. Lee, S.Noh: Block Recycling Schemes and Their Cost-based Optimization in NAND Flash Memory Based Storage System, EMSOFT'07, Sept. 2007

Seiten-/bzw. Sektorabbildung mit *Flash transaction layer* (FTL)



Invertierte Seitentabelle im Flashspeicher gespeichert (Extra Bits); “Normale Seitentabelle” während der Initialisierung erzeugt; Seitentabelle kann sehr groß werden; Wird in kleinen NOR Flash-Speichern benutzt.

Sektor ≈ Seite
\+ Extra Bits

Ausnutzung von Regularität

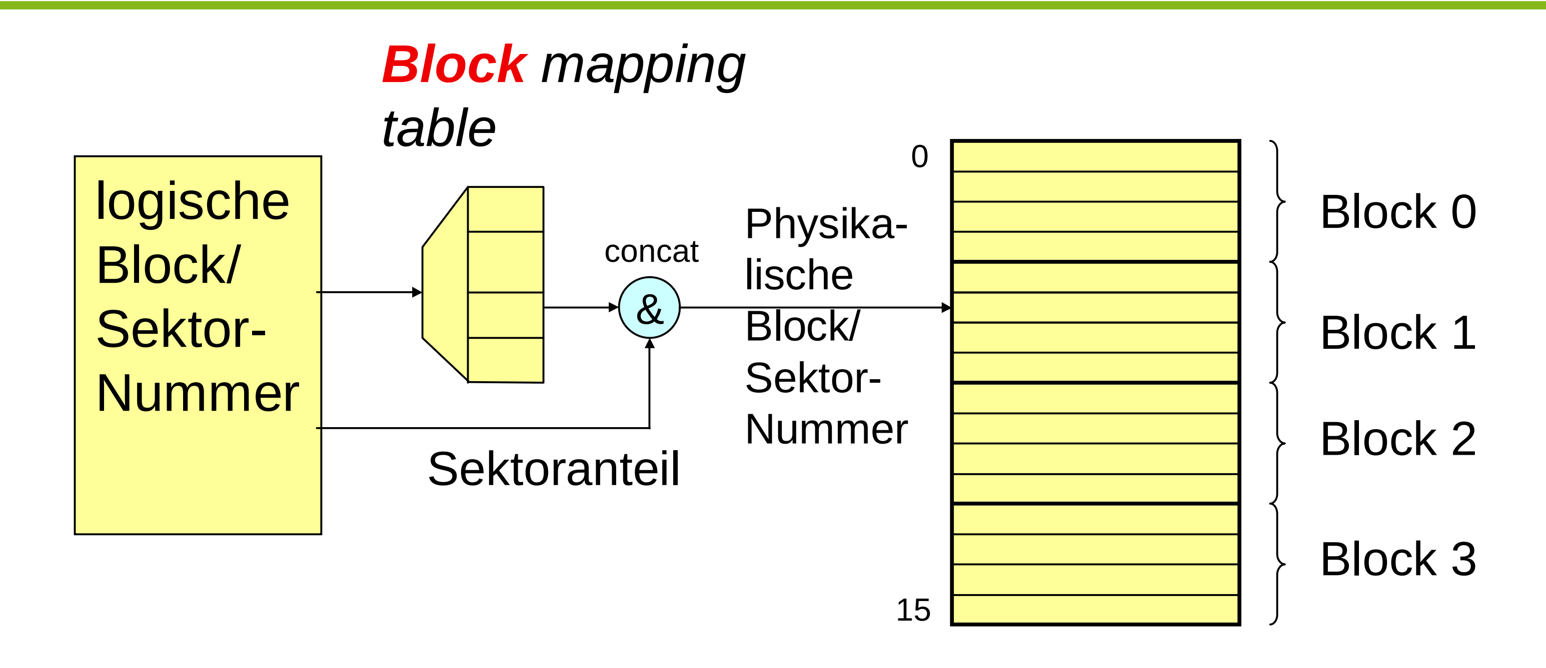
Häufig lange Sequenzen von sequentiellen Schreib-vorgängen

NIKON Trace Access Pattern (Write)

Sector number: 0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000

Write command: 0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000

Block mapping flash transaction layer (FTL)

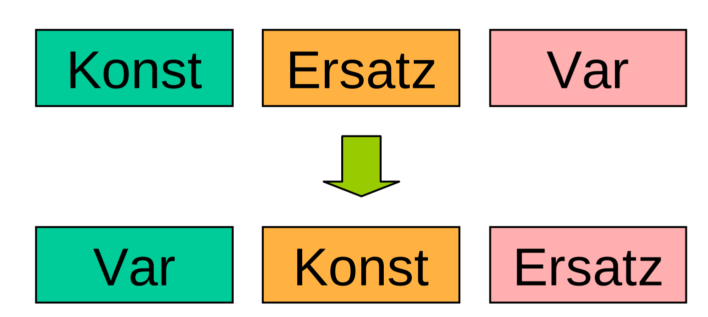


- Abbildungstabellen kleiner als bei Seiten-basierten FTLs
- ☞ In großen NAND Flash-Speichern benutzt
- ☞ Einfache Realisierung,
- Wiederholtes Schreiben erfordert Kopieren auf einen neuen Block
- Schlechte *Performance* bei wiederholtem und zufälligem Schreiben
- Hybride Verfahren

Ausgleich der Abnutzung (*wear levelling*)

Beispiel (Lofgren et al., 2000, 2003):

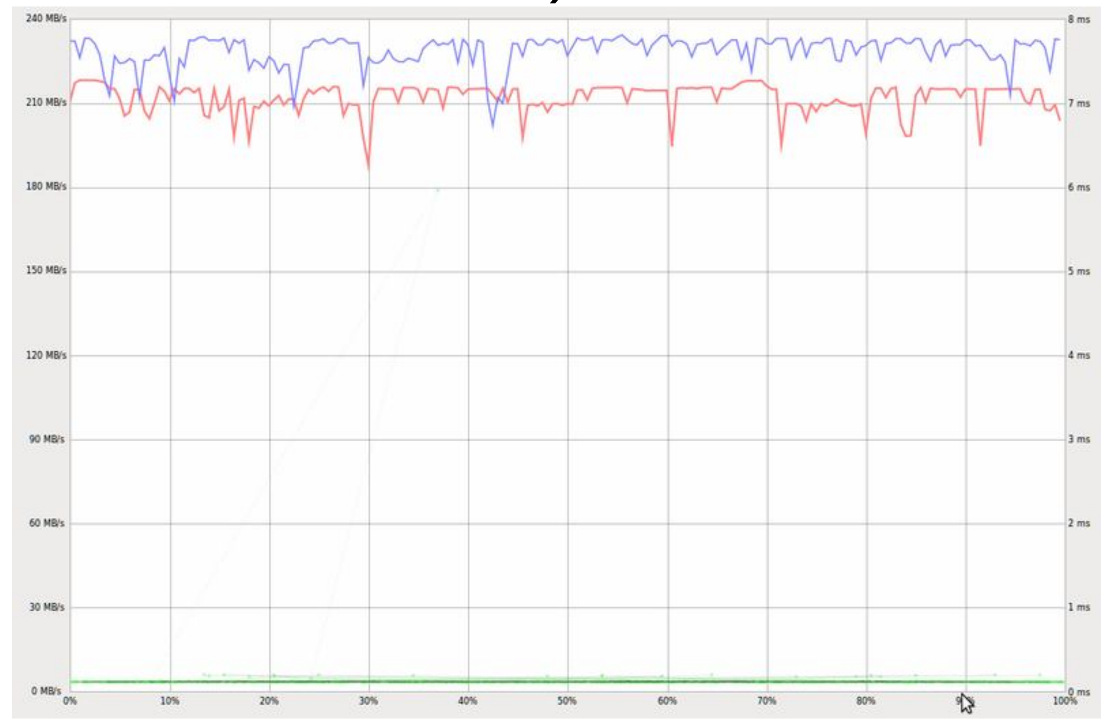
- Jede *erase unit* (Block) besitzt Löschzähler
- 1 Block wird als Ersatz vorgehalten
- Wenn ein häufig genutzter Block frei wird, wird der Zähler gegen den des am wenigsten benutzten Blocks verglichen. Wenn der Unterschied groß ist:
  - Inhalt wenig genutzten Blocks (≈ Konstanten) → Ersatz
  - Inhalt häufig genutzten Blocks → am wenigsten genutzter Block
  - Häufig genutzter Block wird zum Ersatzblock



Source: Gal, Toledo, *ACM Computing Surveys*, June 2005

Solid State Discs (SSDs)

Zugriffszeiten ca. 100x kürzer als HD
Hervorragende Transfergeschwindigkeiten von SSDs
(2013: ~ 500-600 MB/s lesen und schreiben)



http://en.wikipedia.org/wiki/File:480_GB_OCZ-AGIL_ITY3.png

Flashspeicher als „*paging device*“



z/OS FLASH Use Cases

- **Paging**
  - z/OS paging subsystem will work with mix of internal Flash and External Disk
    - Self Tuning based on measured performance
    - Improved Paging Performance, Simplified Configuration
  - Begin Paging 1 MB Large Pages only to Flash
    - Exploit Flash's random I/O read rate to gain CPU performance by enabling additional use of Large Pages. Currently large pages are not pagable.
  - Begin Speculative Page-In of 4K Pages
    - Exploit Flash's random I/O read rate to get Improved Resilience over Disruptions.
    - Market Open, Workload Failover,

Flash-spezifische Dateisysteme

- Zwei Ebenen können ineffizient sein:
  - FTL bildet Magnetplatte nach
  - Standard-Dateisystem basiert auf Magnetplatten

  Beispiel: Gelöschte Sektoren nicht markiert
  ☞ nicht wieder verwendet
- Log-strukturierte Dateisysteme fügen nur neue Informationen zu
  - Für Magnetplatten
    - Schnelle Schreibvorgänge
    - Langsames Lesen (Kopfbewegungen für verteilte Daten)
  - Ideal für Flash-basiertes Dateisystem:
    - Schreibvorgänge in leere Sektoren
    - Lesen nicht langsam, da keine Köpfe bewegt werden

☞ Spezifische log-basierte *Flash*-Dateisysteme
- JFFS2 (NOR)
- YAFFS (NAND)

Source: Gal, Toledo, *ACM Computing Surveys*, June 2005

Vergleich *Harddisc/Flash*-Speicher (2011)

	Flash	HDD
Zugriffszeit (random) [ms]	~0.1	5-10
Kosten [$/GB]	1,2-2 (Fixanteil gering)	0,05-0,1, Fixanteil !
Kapazität [GB]	Typ. < 120	1000-3000
Leistungsaufnahme [W]	Ca. 1/3-1/2 der HDD-Werte	Typ. 12-18, laptops: ~2
Defragmentierung	unwichtig	Zu beachten
Zeitverhalten	Relativ deterministisch	Von Kopfbewegung abhängig
Anzahl der Schreibvorgänge	begrenzt	unbegrenzt
Verschlüsselung	Überschreiben unver-schlüsselter Info schwierig	Überschreiben einfach
Mechanische Empfindlichk.	Gering	Stoßempfindlich
Wiederbenutzung von Blöcken	Erfordert Extra-Löschen	Überschreiben

[wikipedia, 2011]

Vergleich Flash/Microdrive (2002)

	Sandisk Type I Flash	Sandisk Type II Flash	IBM Microdrive DSCM-10340
Kapazität [MB]	64	300	340
El. Leistung [W] (standby/operating)	0,15/0.66	0,15/0,66	0,07/0.83
Schreibzyklen	300.000	300.000	unbegrenzt
Mittl. Abstand zwischen Fehlern [h]	>1.000.000	>1.000.000	*service-life*=min(5J, 8800 h *operating*)
Fehlerraten, unkorrigierbar	< 1 per 10^{14}	<1 per 10^{14}	<1 per 10^{13}
Max. Zahl Einschaltvorgänge	unbegrenzt	unbegrenzt	300.000
Schock-Toleranz	2000 G; 2000 G	2000 G;	175 G; 1500 G

Source: Hennessy/Patterson, Computer Architecture, 2002

Mögliche Stufen der Speicherhierarchie
und derzeit eingesetzte Technologien

Tertiärspeicher (Optische Speicher, Bänder)

↕

Sekundärspeicher (Platten, Flash)

↕

Plattencaches (DRAM, Flash)

↕

Haupt- bzw. Primärspeicher (DRAM)

↕

Caches (SRAM)

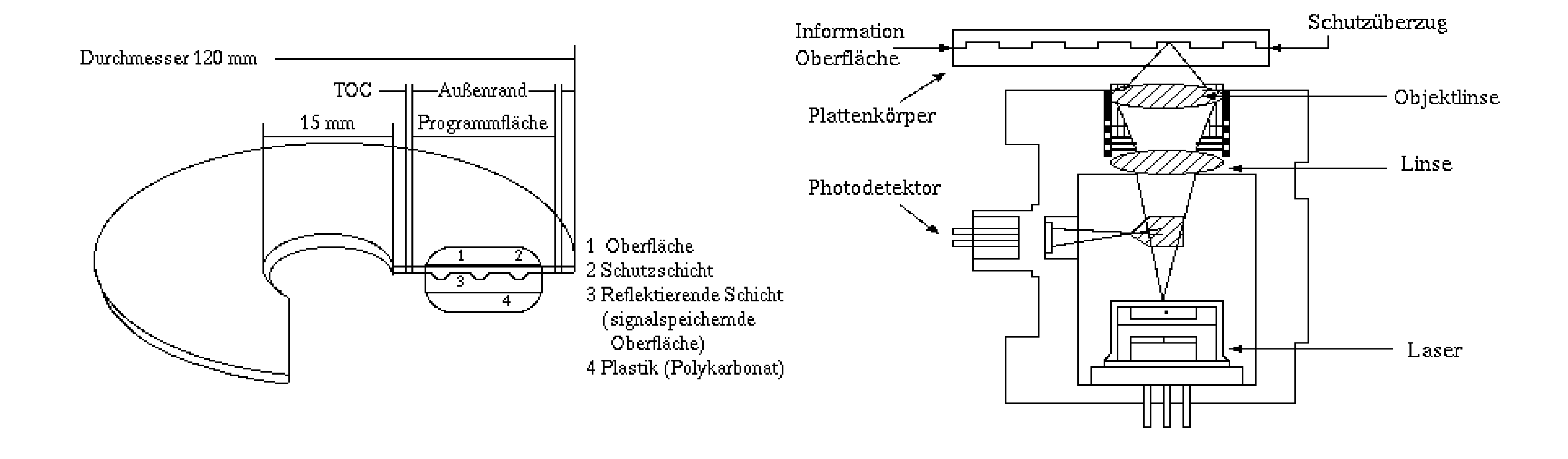
↕

Register (SRAM)

persistent

CD-ROM

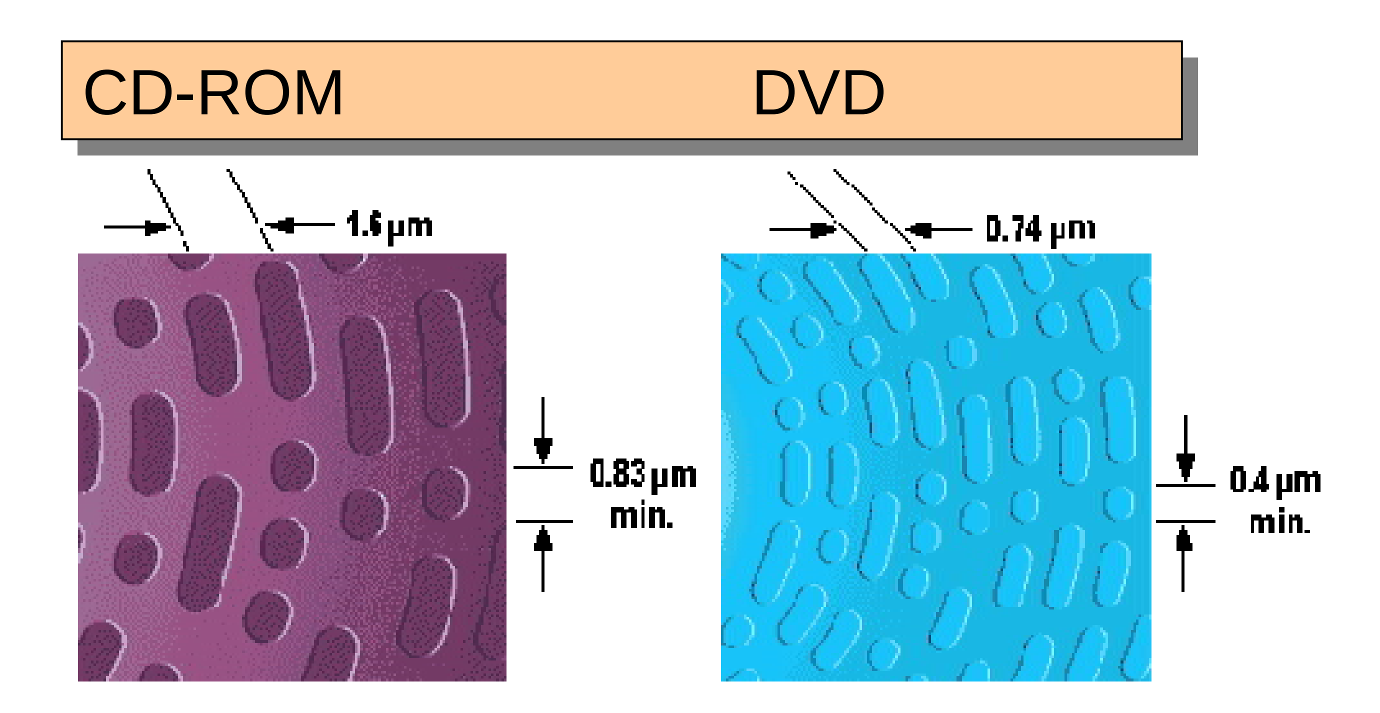
Datenträger auf der Basis der Technik von Audio-CDs. Informationen als Vertiefungen in einer reflektierenden Oberfläche gespeichert. Abtastung mit 780 nm Laser.



DVD-Laufwerke (1)

Kapazität gegenüber CDs erhöht:

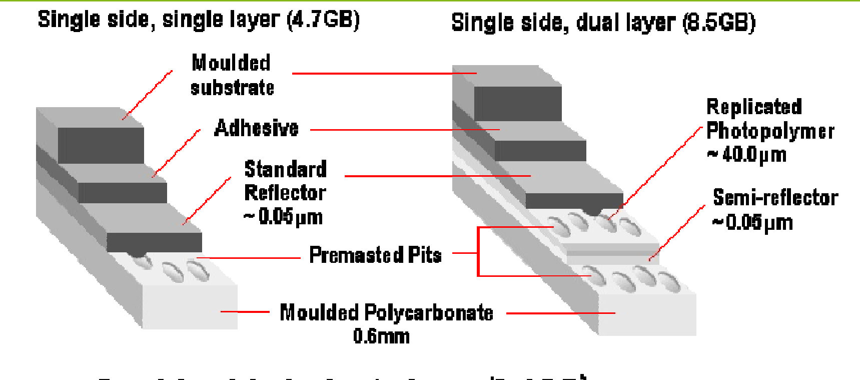
1. Abstände und Größe der Vertiefungen werden reduziert, Abtastung mit 650 nm (rotem) Laserstrahl:

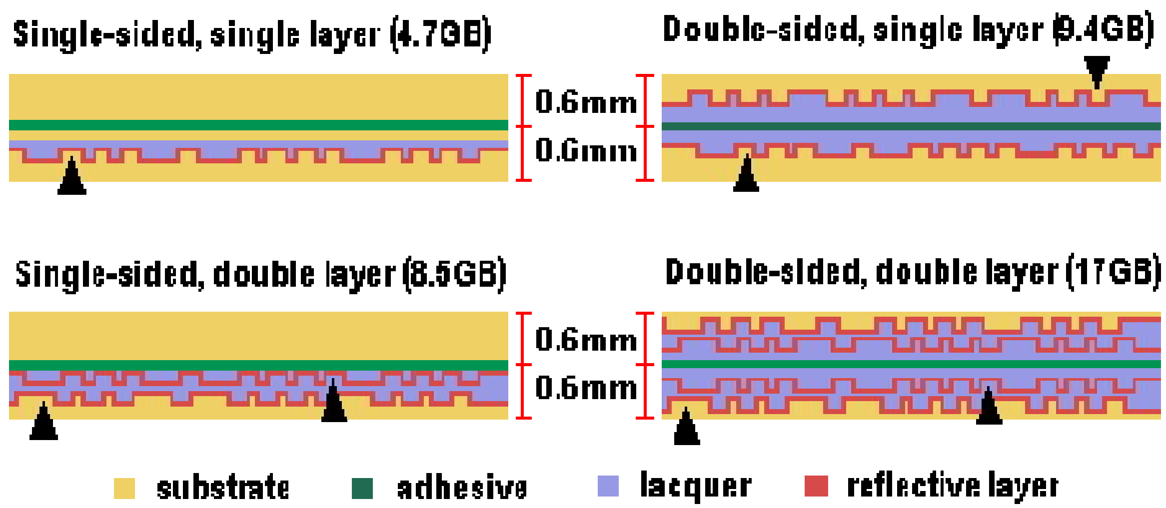


☞ Erhöhung der Kapazität um den Faktor 4.

DVD-Laufwerke (2)

2. Informationen können in zwei Ebenen und auf beiden Seiten gespeichert werden.





Datenrate 1,25 MB/s bei einfacher Geschwindigkeit.

MPEG-2 bei Speicherung von Filmen.

Blu*-ray disc (BD) : Motivation

Blue laser

22 GB is necessary to record BS digital broadcasting.

BD

Red laser

DVD

4.7 GB was needed to record movie in MPEG2 for 2 hours and 15 minutes.

Infrared laser

1.2mm

CD

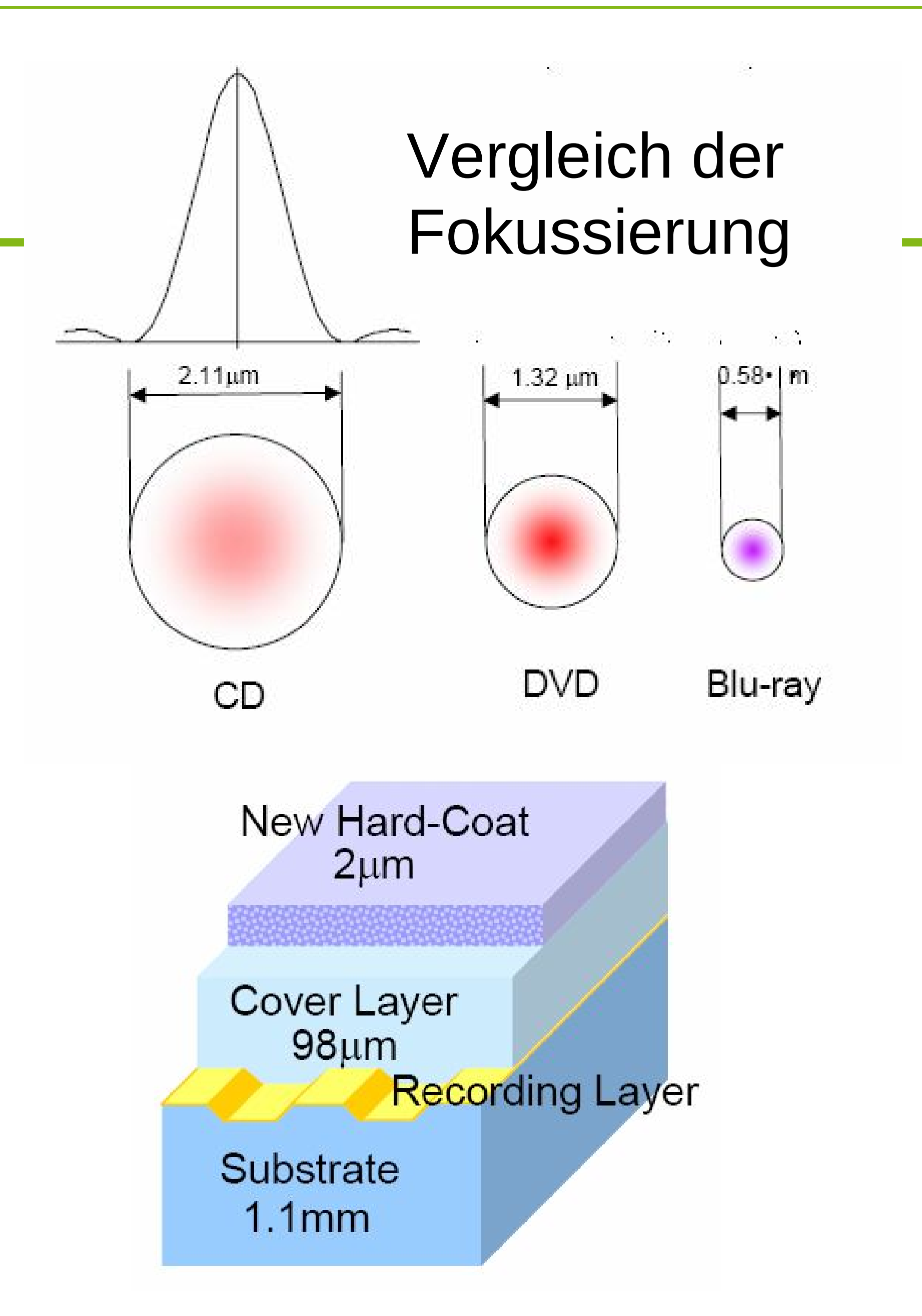
780 MB was needed to record 2ch PCM audio for 74 minutes.

Fig.1.1.1 Evolution of consumer optical discs

*Falsche Schreibweise, damit Eintrag als Markenzeichen möglich

http://www.blu-raydisc.com/assets/downloadablefile/general_bluraydiscformat-12834.pdf

Blu-ray: Eigenschaften



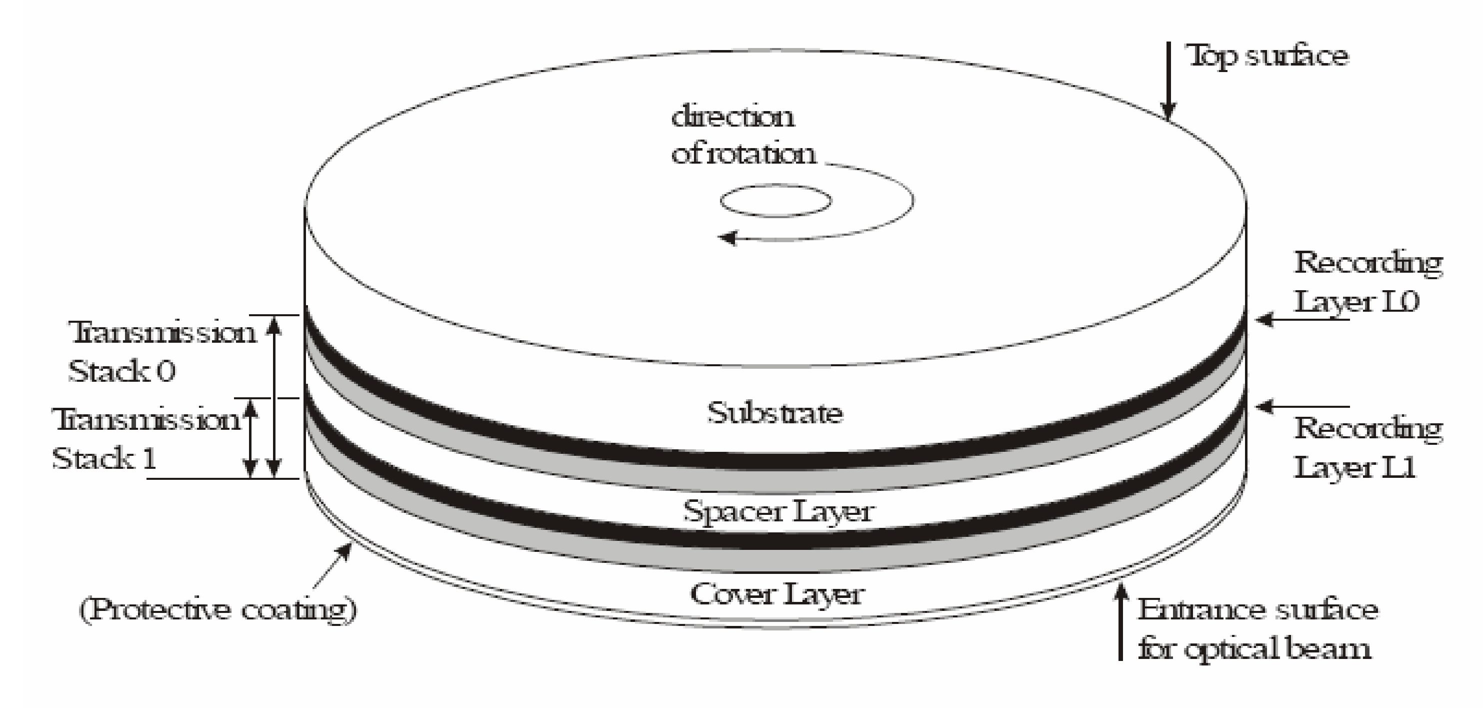
Verkürzung der Wellenlänge auf
405 nm (blau)

Reduktion der Dicke der transparenten Schicht auf 100 µm, zur besseren Fokussierung des Laserstrahls, bessere Fehlerkorrektur; zusätzliche schmutzresistente Schicht

http://www.blu-raydisc.com/assets/downloadablefile/
general_bluraydiscformat-12834.pdf

Blu-ray: Eigenschaften (2)

23.3/25/27 GB in einer Speicherebene
46.6/50/54 GB in zwei Speicherebenen
Experimentell 200 GB



http://www.blu-raydisc.com/assets/downloadablefile/general_bluraydiscformat-12834.pdf

Vergleich Übertragungsgeschwindigkeiten

Annahme: Transport einer Blu-ray, als *Dual Layer* beschrieben (Kapazität=50 GB) auf der Strecke Dortmund - München (Strecke ~ 500km) mit einer Fahrrad-Staffel mit 36km/Std im Mittel. Gleichzeitig DSL-Übertragung mit Übertragungsrate 8 M Bit/s. Wie viel Prozent der maximalen Übertragungsrate muss die DSL-Strecke erreichen, damit nicht die Radfahrer gewinnen ? (Vernachlässigung von Extra-Bits)

$x \times$ Übertragungsrate $\times$ Zeit des Radfahrers = Kapazität

x = Kapazität $\times$ Geschwindigkeit / (Übertragungsrate $\times$ Strecke)

$x = 50 \times 10^9 \times 8 \times 36 / (8 \times 10^6 \times 60 \times 60 \times 500)$

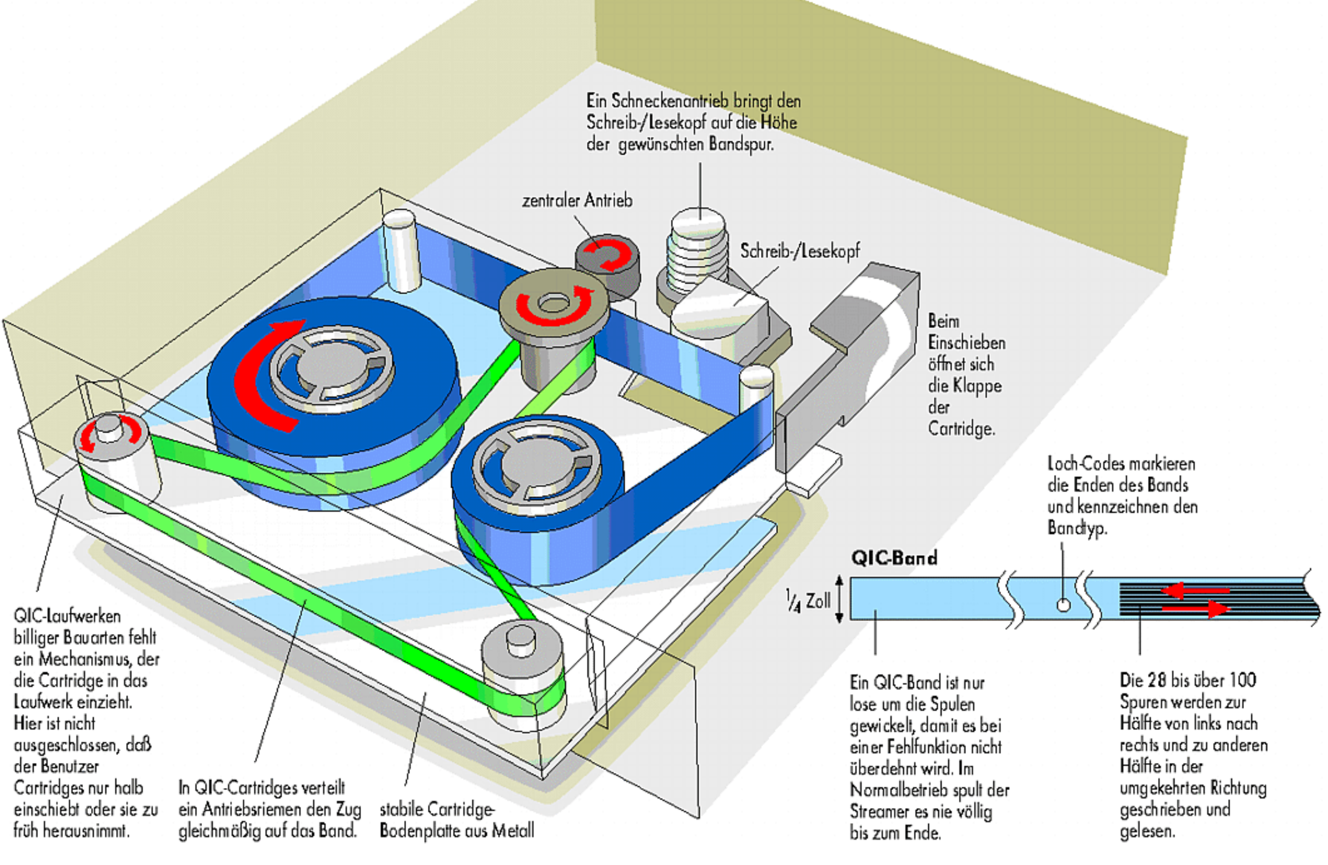
$x = 50 \times 10^9 \times 8 \times 36 / (50 \times 10^9 \times 8 \times 36)$

$x = 1$

Die DSL-Strecke müsste 100 % der max. Übertragungsrate erreichen.

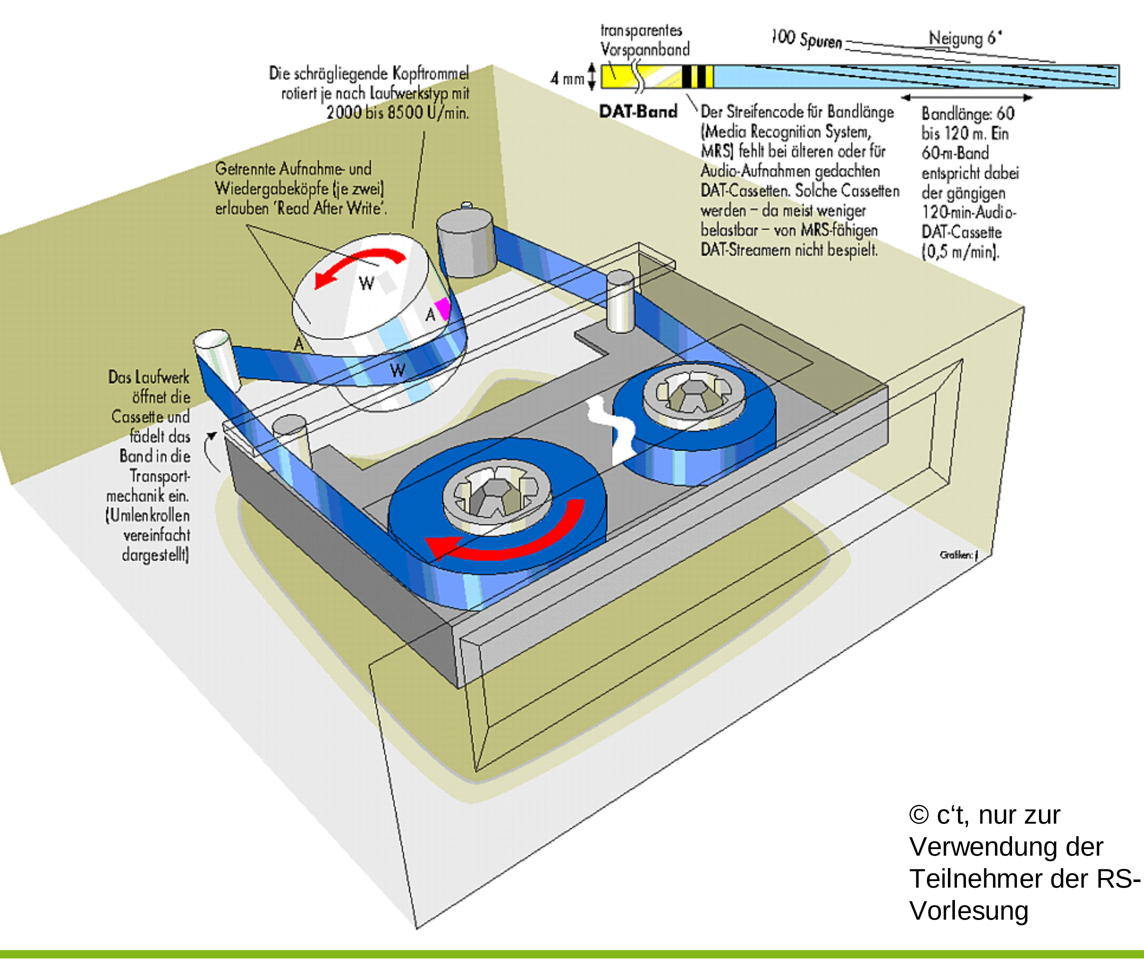
QIC-Laufwerk (*Quarter inch tape*)

Horizontale Aufzeichnung in mehreren (bis zu 100) Spuren nacheinander, optimiert auf *streaming*-Betrieb mit hoher Bandgeschwindigkeit, bis ~5 GB (20 GB bei Weiterentwicklung Travan-5)



DAT-Laufwerk (*digital audio tape*)

Technik der *digital audio tapes,*
Schrägspur-aufzeichnung,
Weiterentwicklung DDS-4; bis 20 GB (40GB komprimiert);



Linear Tape Open (LTO) - 4

[www.wikipedia.de]

Linear Tape Open (LTO) - 4

Übersicht der Generationen								
Laufwerks-Generation	Bandkapazität bei gleicher Generation		maximale Transferrate		kompatible Bänder		Verfügbarkeit	
	ohne Kompression	mit Kompression	ohne Kompression	mit Kompression	Schreiben	Lesen	Lizenzen seit	Produkte seit
Ultrium 1	100 GB [5]	200 GB [5]	20 MB/s	40 MB/s [5]	Ultrium 1	Ultrium 1	1998 [5]	2000 [5]
Ultrium 2	200 GB [5]	400 GB [5]	40 MB/s	80 MB/s [5]	Ultrium 1 Ultrium 2	Ultrium 1 Ultrium 2	April 2002 [5]	Ende 2002 [5]
Ultrium 3	400 GB [5]	800 GB [5]	80 MB/s	160 MB/s [5]	Ultrium 2 Ultrium 3	Ultrium 1 Ultrium 2 Ultrium 3	Juli 2004 [5]	Ende 2004 [5]
Ultrium 4	800 GB [5]	1.600 GB [5]	120 MB/s	240 MB/s [5]	Ultrium 3 Ultrium 4	Ultrium 2 Ultrium 3 Ultrium 4	Ende 2006 [5]	Frühjahr 2007 [6]
Ultrium 5	1.500 GB	3.000 GB [5]	140 MB/s	280 MB/s [5]	Ultrium 4 Ultrium 5	Ultrium 3 Ultrium 4 Ultrium 5	Sommer 2009 [7]	Frühjahr 2010 [8]
Ultrium 6	3.200 GB	8.000 GB [5]	210 MB/s	525 MB/s [5]	-	-	-	-
Ultrium 7	6.400 GB	16.000 GB [5]	315 MB/s	788 MB/s [5]	-	-	-	-
Ultrium 8	12.800 GB	32.000 GB [5]	472 MB/s	1180 MB/s [5]	-	-	-	-

[www.wikipedia.de]

Bewertung von Band-basierten Medien

Lange Suchzeiten, geringe Transferraten.

Abnutzung

- Schrägspur: "Hunderte von Durchläufen"
- Parallelspur: "Tausende bis Millionen von Durchläufen„

Ursprünglich deutlich kostengünstiger als Platten,
da nur das Speichermedium repliziert wird.

Hennessy/Patterson: Gegenwärtiger Kostenvorteil?

(2011): ~22 € für Archivierung von 800 GB über LTO-4; bietet immer noch Kosten- und Archivierungsvorteile

Zusammenfassung



Nicht-flüchtige Speicher:

- Plattenspeicher sind traditionell preisgünstiges Speichermedium
- RAID-Speicher bieten benötigte Zuverlässigkeit
- Flash-Speicher bieten neue Speicheroptionen, trotz begrenzter Anzahl von Schreiboperationen
- Kapazität optischer Speicher steigt weiter
- Unterschiedliche Beurteilung der Preisvorteile von Bandspeichern